

Sybase® PowerDesigner® Object Oriented Model

User's Guide

Version 9.5
38086-01-0950-01
Last modified: July 2002

Copyright © 2002 Sybase, Inc. All rights reserved.
Information in this manual may change without notice and does not represent a commitment on the part of Sybase, Inc. and its subsidiaries.
Sybase, Inc. provides the software described in this manual under a Sybase License Agreement. The software may be used only in accordance with the terms of the agreement.
No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.
Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, SYBASE (logo), AccelaTrade, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-Translator, APT-Library, ASEP, Backup Server, BayCam, Bit-Wise, BizTracker, Certified PowerBuilder Developer, Certified SYBASE Professional, Certified SYBASE Professional Logo, ClearConnect, Client-Library, Client Services, CodeBank, Column Design, ComponentPack, Connection Manager, Convoy/DM, Copernicus, CSP, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, e-ADK, E-Anywhere, e-Biz Integrator, E-Whatever, EC-GATEWAY, ECMAP, ECRTP, eFulfillment Accelerator, Electronic Case Management, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, eremote, Everything Works Better When Everything Works Together, EWA, Financial Fusion, Financial Fusion Server, Formula One, Gateway Manager, GeoPoint, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InstaHelp, InternetBuilder, iremote, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Logical Memory Manager, MainframeConnect, Maintenance Express, Manage Anywhere Studio, MAP, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, MethodSet, ML Query, MobiCATS, MySupport, Net-Gateway, Net-Library, New Era of Networks, Next Generation Learning, Next Generation Learning Studio, O DEVICE, OASiS, OASiS logo, ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Biz, Open Business Interchange, Open Client, Open ClientConnect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, Partnerships that Work, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, PhysicalArchitect, Pocket PowerBuilder, PocketBuilder, Power++, Power Through Knowledge, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, Powering the New Economy, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Rapport, Relational Beans, Report Workbench, Report-Execute, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Resource Manager, RW-DisplayLib, RW-Library, SAFE, SAFE/PRO, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL SMART, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Server SNMP SubAgent, SQL Station, SQLJ, Stage III Engineering, Startup.Com, STEP, SupportNow, S.W.I.F.T. Message Format Libraries, Sybase Central, Sybase Client/Server Interfaces, Sybase Development Framework, Sybase Financial Server, Sybase Gateways, Sybase Learning Connection, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase Synergy Program, Sybase Virtual Server Architecture, Sybase User Workbench, SybaseWare, Syber Financial, SyberAssist, SybMD, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, The Enterprise Client/Server Company, The Extensible Software Platform, The Future Is Wide Open, The Learning Connection, The Model For Client/Server Solutions, The Online Information Center, The Power of One, TradeForce, Transact-SQL, Translation Toolkit, Turning Imagination Into Reality, UltraLite, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Versacore, Viewer, VisualWriter, VQL, WarehouseArchitect, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server and XP Server are trademarks of Sybase, Inc. or its subsidiaries.

All other trademarks are property of their respective owners.

Contents

About This Book.....xix

1 Object-Oriented Model Basics 1
Functional overview 2
UML and object-oriented modeling..... 3
What is an OOM? 4
Working with diagrams in an OOM 9
Objects in an OOM..... 10
Defining an OOM 16
Object language overview..... 16
Extended dependencies..... 16
Creating an OOM 17
Selecting extended model definitions at model creation .. 21
Changing the object language of an OOM..... 22
Opening an existing OOM..... 23
Detaching an OOM from the Workspace..... 24
Defining OOM properties..... 24
Previewing the code of an OOM 26
Modifying the code using Preview..... 27
Defining OOM model options..... 28
Defining packages in an OOM..... 32
OOM Package properties..... 32
Previewing the code of a package in an OOM..... 33
Defining the diagram type of a new package 34
Modifying package display preferences 34

2 Building a Use Case Diagram.....35
Use Case basics..... 36
Defining a use case diagram..... 36
Why build a use case diagram?..... 36
Creating a use case diagram 37
Defining use cases 38
Use case properties 38

Creating a use case 41
Modifying use case properties..... 43
Modifying use case display preferences 43
Defining actors in a use case diagram 44
Actor properties in a use case diagram..... 45
Creating an actor in a use case diagram 47
Showing the symbol of an actor 48
Modifying actor properties in a use case diagram..... 49
Modifying actor display preferences..... 49
Defining associations in a use case diagram 50
Association properties in a use case diagram..... 51
Creating an association in a use case diagram 51
Modifying association properties in a use case diagram.. 53
Modifying association display preferences in a use case diagram 53
Defining generalizations in a use case diagram 54
Generalization properties in a use case diagram..... 55
Creating a generalization in a use case diagram 56
Modifying generalization properties in a use case diagram 57
Modifying generalization display preferences..... 58
Defining dependencies in a use case diagram..... 59
Dependency properties in a use case diagram..... 59
Creating a dependency in a use case diagram..... 60
Modifying dependency properties in a use case diagram 62
Modifying dependency display preferences 62

3 Building a Class Diagram..... 63
Class diagram basics..... 64
Defining a class diagram..... 64
Why build a class diagram? 64
Creating a class diagram 64
Defining classes..... 65
Class properties 66
Creating a class..... 70
Creating a class from an interface 71
Defining Java BeanInfo classes..... 72
Creating Java BeanInfo classes..... 74
Classifiers..... 77
Defining inner classifiers 77
Adding objects to a class 81
Attaching a classifier to a data type or a return type 83
Previewing the code of a class or interface..... 89
Viewing the migrated attributes of a class..... 91
Customizing scripts for classes and interfaces..... 93
Modifying class properties..... 94

Modifying class display preferences 95
Defining interfaces 96
Interface properties 97
Creating an interface 99
Creating an interface from a class 100
Adding objects to an interface 101
Modifying interface properties 104
Modifying interface display preferences 104
Defining attributes in a class diagram 105
Attribute properties in a class diagram 105
Creating an attribute in a class diagram 109
Attaching an attribute to a domain in a class diagram ... 110
Adding Getter and Setter operations to a classifier 111
Copying an attribute to another class 113
Modifying attribute properties in a class diagram 114
Modifying attribute display preferences in a class diagram 114
Defining identifiers 116
Identifier properties 116
Creating a primary identifier 117
Creating an identifier 118
Adding attributes to an identifier 118
Modifying identifier properties 119
Defining operations 120
Operation properties 120
Defining operation parameters 122
Creating an operation 125
Adding operations to a class 126
Creating an implementation operation 133
Modifying the code of an implementation operation 135
Copying an operation to another class 136
Modifying operation properties 137
Modifying operation display preferences 138
Defining associations in a class diagram 139
Association properties in a class diagram 139
Association role in a class diagram 140
Creating an association in a class diagram 143
Creating an association class 144
Migrating association roles in a class diagram 147
Rebuilding data type links 149
Modifying association properties in a class diagram 150
Modifying association display preferences in a class diagram 150
Defining generalizations in a class diagram 151
Generalization properties in a class diagram 152
Creating a generalization in a class diagram 152

Modifying generalization properties in a class diagram.. 154
Modifying generalization display preferences..... 155
Defining dependencies in a class diagram 156
Dependency properties in a class diagram 157
Creating a dependency in a class diagram 158
Modifying dependency properties in a class diagram 160
Modifying dependency display preferences 160
Defining realizations..... 161
Realization properties..... 161
Creating a realization 162
Modifying realization properties..... 163
Modifying realization display preferences..... 164
Defining domains in an OOM..... 165
Domain properties in an OOM 165
Creating a domain in an OOM 166
Selecting a data type for a domain in an OOM 167
Updating attributes using a domain in an OOM 169
Defining check parameters..... 170
Setting standard check parameters for OOM objects.... 170
Defining additional check parameters for OOM objects. 171

4 Building an Object diagram 173
Object diagram basics..... 174
Defining an object diagram 174
Why build an object diagram?..... 174
Creating an object diagram 175
Defining objects in an object diagram..... 176
Object properties in an object diagram 177
Creating an object in an object diagram..... 178
Modifying object properties in an object diagram..... 179
Modifying object display preferences 180
Defining attribute values 181
Attribute value properties 182
Creating an attribute value 183
Modifying attribute value properties 184
Defining instance links in an object diagram 185
Instance link properties in an object diagram..... 186
Creating an instance link in an object diagram 187
Modifying instance link properties in an object diagram. 188
Modifying instance link display preferences..... 189
Defining dependencies in an object diagram..... 190
Dependency properties in an object diagram..... 190
Creating a dependency in an object diagram..... 190
Modifying dependency properties in an object diagram. 192
Modifying dependency display preferences 193

5 Building a Collaboration Diagram.....195
Collaboration diagram basics 196
Defining a collaboration diagram..... 196
Why build a collaboration diagram? 196
Creating a collaboration diagram 197
Converting a collaboration diagram to a sequence diagram 197
Defining actors in a collaboration diagram..... 199
Actor properties in a collaboration diagram..... 199
Creating an actor in a collaboration diagram..... 200
Showing the symbol of an actor 201
Modifying actor properties in a collaboration diagram.... 201
Modifying actor display preferences..... 202
Defining objects in a collaboration diagram 203
Object properties in a collaboration diagram..... 204
Creating an object in a collaboration diagram 204
Modifying object properties in a collaboration diagram .. 206
Modifying object display preferences 206
Defining instance links in a collaboration diagram..... 207
Instance link properties in a collaboration diagram 209
Creating an instance link in a collaboration diagram..... 210
Modifying instance link properties in a collaboration diagram 211
Modifying instance link display preferences 212
Defining messages in a collaboration diagram 213
Defining message properties in a collaboration diagram 215
Creating a message in a collaboration diagram..... 217
Creating a sequence number in a collaboration diagram 219
Managing sequence numbers in a collaboration diagram 219
Modifying message properties in a collaboration diagram 222
Modifying message display preferences in a collaboration diagram..... 222

6 Building a Sequence Diagram.....223
Sequence diagram basics 224
Defining a sequence diagram..... 224
Why build a sequence diagram?..... 225
Creating a sequence diagram 225
Converting a sequence diagram to a collaboration diagram 226
Defining actors in a sequence diagram 227
Actor properties in a sequence diagram 228

Creating an actor in a sequence diagram 229
Showing the symbol of an actor 229
Modifying actor properties in a sequence diagram 230
Modifying actor display preferences 230
Defining objects in a sequence diagram 231
Object properties in a sequence diagram 232
Creating an object in a sequence diagram 233
Modifying object properties in a sequence diagram 234
Modifying object display preferences in a sequence diagram 235
Defining messages in a sequence diagram 236
Defining message properties in a sequence diagram 238
Creating a message in a sequence diagram 242
Creating a Create message in a sequence diagram 243
Creating Destroy messages in a sequence diagram 244
Creating a recursive message in a sequence diagram 246
Creating an activation 249
Managing activations 250
Modifying message properties in a sequence diagram 253
Modifying message display preferences in a sequence diagram 254

7 Building a Statechart diagram 255
Statechart diagram basics 256
Defining a statechart diagram 256
Why build a statechart diagram? 256
Statechart diagrams vs sub-statechart diagrams 257
Defining a default classifier in the statechart diagram 258
Creating a statechart diagram 259
Converting a statechart diagram to a composite state 259
Defining starts in a statechart diagram 261
Start properties in a statechart diagram 261
Creating a start in a statechart diagram 262
Modifying start properties in a statechart diagram 263
Modifying start display preferences 263
Defining states 264
State properties 264
Defining composite states 266
Creating a state 267
Creating a composite state 268
Opening a composite state 270
Closing a composite state 270
Modifying state properties 270
Modifying state display preferences 271
Defining actions 272
Action properties 273

Creating an action on a state 275
Defining an action on a transition 276
Modifying action properties 277
Defining transitions in a statechart diagram 278
Transition properties in a statechart diagram 279
Creating a transition in a statechart diagram 282
Modifying transition properties in a statechart diagram 283
Modifying transition display preferences in a statechart diagram 284
Defining events 285
Event properties 286
Defining event parameters 286
Defining event arguments 289
Creating an event 289
Modifying event properties 290
Defining junction points 291
Junction point properties 291
Creating a junction point 291
Modifying junction point properties 293
Modifying junction point display preferences 293
Defining ends in a statechart diagram 294
End properties in a statechart diagram 294
Creating an end in a statechart diagram 294
Modifying end properties in a statechart diagram 296
Modifying end display preferences 296

8 Building an Activity Diagram 297
Activity diagram basics 298
Defining an activity diagram 298
Why build an activity diagram? 299
Activity diagrams vs sub-activity diagrams 300
Creating an activity diagram 301
Converting an activity diagram to a composite activity 301
Defining starts in an activity diagram 303
Start properties in an activity diagram 303
Creating a start in an activity diagram 304
Modifying start properties in an activity diagram 305
Modifying start display preferences 305
Defining activities 306
About the UML notation 306
Activity properties 307
Defining composite activities 309
Creating an activity 310
Creating a composite activity 312
Opening a composite activity 314
Closing a composite activity 314

Modifying activity properties 314
Modifying activity display preferences 315
Defining object states 316
Object state properties 317
Creating an object state 319
Modifying object state properties 320
Modifying object state display preferences 320
Defining organization units in an activity diagram 321
Organization unit properties in an activity diagram 322
Creating an organization unit in an activity diagram 323
Attaching an activity to an organization unit 327
Detaching an activity from an organization unit 327
Managing swimlanes in an activity diagram 328
Modifying organization unit properties in an activity diagram 331
Modifying organization unit display preferences in an activity diagram 331
Defining transitions in an activity diagram 332
Transition properties in an activity diagram 333
Creating a transition in an activity diagram 335
Modifying transition properties in an activity diagram 336
Modifying transition display preferences in an activity diagram 337
Defining decisions in an activity diagram 338
Defining conditional branches 339
Defining merges 339
Decision properties in an activity diagram 340
Creating a decision in an activity diagram 341
Modifying decision properties in an activity diagram 342
Modifying decision display preferences in an activity diagram 343
Defining synchronizations in an activity diagram 344
Defining forks 344
Defining joins 344
Synchronization properties in an activity diagram 345
Creating a synchronization in an activity diagram 346
Drawing the synchronization symbol in an activity diagram 347
Modifying synchronization properties in an activity diagram 348
Defining ends in an activity diagram 349
End properties in an activity diagram 349
Creating an end in an activity diagram 350
Modifying end properties in an activity diagram 351
Modifying end display preferences 352

9 Building a Component Diagram 353
Component diagram basics 354
Defining a component diagram 354
Why build a component diagram? 354
Creating a component diagram 355
Defining components 356
Component properties 357
Defining interfaces for a component 359
Defining classes for a component 361
Defining file objects for a component 362
Viewing operations of a component 363
Creating a class diagram for a component 364
Creating a component 365
Deploying a component to a node 367
Modifying component properties 368
Modifying component display preferences 369
Defining generalizations in a component diagram 370
Generalization properties in a component diagram 370
Creating a generalization in a component diagram 371
Modifying generalization properties in a component diagram 372
Modifying generalization display preferences 373
Defining dependencies in a component diagram 374
Dependency properties in a component diagram 374
Creating a dependency in a component diagram 375
Modifying dependency properties in a component diagram 376
Modifying dependency display preferences 377

10 Building a Deployment Diagram 379
Deployment diagram basics 380
Defining a deployment diagram 380
Why build a deployment diagram? 380
Creating a deployment diagram 381
Defining nodes 382
Defining node diagrams 382
Node properties 384
Creating a node 384
Modifying node properties 386
Modifying node display preferences 387
Defining component instances 388
Component instance properties 389
Creating a component instance 389
Modifying component instance properties 391
Modifying component instance display preferences 391

Defining file objects in a deployment diagram 392
File object properties in a deployment diagram 392
Creating a file object in a deployment diagram 393
Defining node associations 396
Node association properties 396
Creating a node association 397
Modifying node association properties 399
Modifying node association display preferences 400
Defining dependencies in a deployment diagram 401
Dependency properties in a deployment diagram 401
Creating a dependency in a deployment diagram 402
Modifying dependency properties in a deployment diagram 403
Modifying dependency display preferences 404

11 Working with Platform Dependent Components 405
Platform overview 406
Working with the Java family 406
Working with the .NET family 406
Defining EJBs 408
Using EJB types 409
EJB properties 410
Creating an EJB with the wizard 412
Understanding EJB support in an OOM 416
Using the EJB deployment descriptor 422
Defining interfaces and classes for EJBs 423
Defining operations for EJBs 426
Generating EJBs 429
Generating .JAR files 438
Reverse engineering EJBs 439
Defining servlets 445
Servlet page of the component 445
Defining servlet classes 446
Creating a servlet with the wizard 446
Understanding servlet initialization and synchronization 448
Generating servlets 448
Reverse engineering servlets 452
Defining JSPs 454
JSP page of the component 454
Defining file objects for JSPs 455
Creating a JSP with the wizard 455
Generating JSPs 457
Reverse engineering JSPs 460
Defining ASP.NET 462
ASP page of the component 462
Defining file objects for ASP.NET 463

Creating an ASP.NET with the wizard 464
Generating ASP.NET 466

12 Working with Web Services 469
Defining Web services 470
Web services overview 471
Defining Web services targets 473
Defining Web service component properties 473
Creating a Web service with the wizard 476
Using data types with Web services 478
Defining Web service classes 480
Detail page of the class 480
Preview page of the class 480
Defining Web service operation properties 481
Defining Web service component instances 482
Web Service page of the component instance 482
WSDL page of the component instance 483
Using node properties 484
Generating Web services 485
Generating server side code for .NET 485
Generating server side code for Java 488
Generating Web services 490
Creating Web service proxy components 492
Generating client side code 492
Reverse engineering Web services 493
Importing a WSDL 493
Browsing WSDL from UDDI 496

13 Working with Object-Oriented Models 501
Defining related diagrams 502
Using related diagrams 502
Adding related diagrams 503
Opening related diagrams 504
Linking objects to other objects 505
Linking a class or an interface to a use case 506
Linking a class or an interface to an actor 507
Linking a classifier to a state 510
Linking a class to an object 511
Linking a component to a component instance 511
Linking a class to an object state 512
Linking an object to an object state 513
Linking a message to an operation 514
Linking an association to an instance link 515
Checking an OOM 516
OOM check options 516

Object selection in the Check Model 517
Checking an OOM 518
Displaying previously applied check options 520
Making corrections based on OOM check results 520
Objects verified during Check model 523
Domain check 523
File object check 523
Replication check 524
Data source check 524
Package check 525
Actor check 526
Actor message check 527
Use case check 527
Class check 527
Identifier check 535
Interface check 536
Class Attribute check 538
Interface Attribute check 540
Class Operation check 541
Interface Operation check 543
Realization check 544
Class Generalization check 544
Interface Generalization check 545
Object check 545
Instance link check 546
Message check 546
State check 547
State action check 548
Event check 549
Junction point check 550
Activity check 550
Decision check 551
Object state check 552
Organization unit check 553
Start check 553
End check 554
Synchronization check 555
Transition check 555
Component check 556
Node check 557
Component instance check 558
Mapping objects in an OOM 560
Understanding O/R mapping 560
Defining data sources in an OOM 561
Defining O/R mapping 564
Mapping a class 565

Mapping an operation ... 570
Mapping an association in a class diagram ... 571
Opening a Rose model in an OOM ... 574
Importing general objects ... 575
Importing use case diagrams ... 575
Importing class diagrams ... 577
Importing collaboration diagrams ... 580
Importing sequence diagrams ... 581
Importing statechart diagrams ... 582
Importing activity diagrams ... 584
Importing component diagrams ... 586
Importing deployment diagrams ... 587

14 Reverse Engineering ... 589
What is reverse engineering? ... 590
Reverse engineering Java ... 591
Reverse engineering Java formats ... 591
Reverse engineering Java options ... 592
Reverse engineering Java files ... 593
Loading a JDK library model in the workspace ... 599
Reverse engineering comments using Javadoc ... 600
Synchronizing a model with generated files ... 604
Reverse engineering PowerBuilder ... 607
Reverse engineering PowerBuilder options ... 607
Loading a PowerBuilder library model in the workspace 608
Reverse engineering objects from a PowerBuilder application ... 609
Reverse engineering objects from SRU files ... 611
Reverse engineering XML ... 613
Reverse engineering XML options ... 614
Reverse engineering XML files ... 615
Reverse engineering into a new OOM ... 617

15 Generating for a Language ... 619
Generation options for languages ... 620
Selecting objects to generate ... 621
Defining generation options ... 622
Defining generation tasks ... 622
Defining generation targets (XEM) ... 623
Generating for C++ ... 626
C++ implementation details ... 626
Managing bidirectional associations ... 626
Generating C++ files ... 628
Generating for C# ... 630
Selecting C# generation options ... 630

Selecting C# generation tasks 630
Generating C# files..... 630
Generating for Java 633
Generating comments using Javadoc..... 633
Selecting Java generation options 635
Selecting Java generation tasks 637
Generating Java files..... 638
Generating for PowerBuilder 640
Selecting PowerBuilder generation options..... 640
Selecting PowerBuilder objects to generate..... 641
Generating for VB.NET..... 643
Selecting VB.NET generation options..... 643
Selecting VB.NET generation tasks..... 643
Generating VB.NET files 643
Generating for XML 646
Generating for XML-Schema 646
Generating for XML-DTD 646
Generating for XML-Data..... 647
Selecting XML generation options..... 647
Generating XML files..... 647

16 Generating from an Object-Oriented Model..... 651
Generation basics..... 652
Target models parameters..... 652
Object selection parameters 654
Generating an Object-Oriented Model from an Object-Oriented Model..... 655
Why generating an OOM into an OOM?..... 655
Generating and updating an OOM 655
Generating a Conceptual Data Model from an Object-Oriented Model..... 662
Generating CDM objects..... 662
Translating OOM to CDM data types 663
Managing object persistence during OOM to CDM generation 663
Generating and updating a CDM..... 668
Generating a Physical Data Model from an Object-Oriented Model..... 675
Generating PDM objects..... 675
Translating OOM to PDM data types 676
Managing object persistence during OOM to PDM generation 677
Generating and updating a PDM..... 684

17 **Using Business Rules in an OOM 695**
What is a business rule in an OOM? 696
Defining business rules in an OOM 697
Business rule properties in an OOM 697
Types of business rule in an OOM 697
Creating a business rule in an OOM 698
Applying business rules to OOM objects 700
Applying a business rule to an OOM object 700
Attaching an expression to a business rule in an OOM . 701

OOM Glossary 703

Index 709

About This Book

Subject

This book describes the PowerDesigner Object-Oriented Model environment. It shows you how to do the following:

- Build an Object-Oriented Model (OOM) using the UML diagrams
- Use modeling elements in diagrams
- Verify the model, merge models and import a Rose model
- Reverse engineer PowerBuilder objects, EJB, Java, and XML files
- Generate files for Java, PowerBuilder, XML, C++, Visual Basic and other languages
- Generate a Conceptual Data Model (CDM), a Physical Data Model (PDM), or an Object-Oriented Model from the OOM
- Generate EJB ™, JSP ™, servlets, ASP.NET
- Create Web services in Java and .NET platforms

Audience

Anyone who will be designing or building an OOM with PowerDesigner Object-Oriented Model will find this book useful. It requires an understanding of object modeling, as well as familiarity with UML and SQL. Some experience with programming languages is helpful. For more information, see the bibliography at the end of this chapter.

Terminology

Most of the terms used along the Object-Oriented Model come from the Unified Modeling Language (UML), defined by the Object Management Group, Inc . (OMG). It is a visual modeling language developed in the mid 90's in an effort to summarize and simplify the large number of object oriented methods that existed at that time. In this manual, we refer to version 1.3 of the UML specification, with some updates from version 1.4.

Documentation primer

The PowerDesigner modeling environment supports several types of models:

- **Conceptual Data Model (CDM)** to model the overall logical structure of a database, independent from any software or data storage structure considerations
- **Physical Data Model (PDM)** to model the overall physical structure of a database, taking into account DBMS software or data storage structure considerations
- **Object Oriented Model (OOM)** to model a software system using an object-oriented approach for Java or other object languages
- **Business Process Model (BPM)** to model the means by which one or more processes are accomplished in operating business practices
- **Free Model (FEM)** to create any kind of chart, diagram, in a context-free environment

This book only explains how to use the OOM. For information on other models or aspects of PowerDesigner, consult the following books:

General Features Guide To get familiar with the PowerDesigner interface before learning how to use any of the models.

Conceptual Data Model Getting Started To learn the basics of the CDM.

Conceptual Data Model User's Guide To work with the CDM.

Physical Data Model Getting Started To learn the basics of the PDM.

Physical Data Model User's Guide To work with the PDM.

Object Oriented Model Getting Started To learn the basics of the OOM.

Business Process Model Getting Started To learn the basics of the BPM.

Business Process Model User's Guide To work with the BPM.

Reports User's Guide To create reports for any or all models.

Repository Getting Started To learn the basics of the Repository.

Repository User's Guide To work in a multi-user environment using a central repository.

Typographic conventions

PowerDesigner documentation uses specific typefaces to help you readily identify specific items:

- `monospace text (normal and` **`bold`**`)`
  Used for: Code samples, commands, compiled functions and files, references to variables.
  Example: `declare user_defined…,` the **`BeforeInsertTrigger`** template.
- UPPER CASE
  Object codes, reversed objects, file names + extension.
  Example: The AUTHOR table appears in the Browser. Open the file OOMAFTER.OOM.
- **bold text**
  Any new term.
  Example: A **shortcut** has a target object.
- SMALL CAPS
  Any key name.
  Example: Press the ENTER key.

Bibliography

James Rumbaugh, Ivar Jacobson, Grady Booch – *The Unified Modeling Language Reference Manual* – Addison Wesley, 1999

Grady Booch, James Rumbaugh, Ivar Jacobson – *The Unified Modeling Language User Guide* – Addison Wesley, 1999

Ivar Jacobson, Grady Booch, James Rumbaugh – *The Unified Software Development Process* – Addison Wesley, 1999

Doug Rosenberg, Kendall Scott – *Use Case Driven Object Modeling With UML A Practical Approach* – Addison Wesley, 1999

Michael Blaha, William Premerlani – *Object-Oriented Modeling and Design for Database Applications* – Prentice Hall, 1998

Geri Schneider, Jason P. Winters, Ivar Jacobson – *Applying Use Cases: A Practical Guide* – Addison Wesley, 1998

Pierre-Alain Muller – *Instant UML* – Wrox Press Inc, 1997

Bertrand Meyer – *Object-Oriented Software Construction* – Prentice Hall, 2nd Edition, 1997

Martin Fowler, Kendall Scott – *UML Distilled Applying The Standard Object Modeling Language* – Addison Wesley, 1997

CHAPTER 1

Object-Oriented Model Basics

About this chapter

This chapter presents the PowerDesigner Object-Oriented Model. It provides you with an introduction to the basic notions of object-oriented modeling and an overview of the Unified Modeling Language (UML).

Contents

Topic	Page
Functional overview	2
UML and object-oriented modeling	3
What is an OOM?	4
Defining an OOM	16
Defining OOM model options	28
Defining packages in an OOM	32

Functional overview

PowerDesigner Object-Oriented Model is a powerful design tool for graphical object-oriented design implementation.

With this product, you can:

- Build an Object-Oriented Model (OOM) using the **use case**, **class, object, collaboration, sequence**, **statechart, activity**, **component** or **deployment** diagrams
- Generate Java, XML files, Sybase PowerBuilder objects
- Generate code for C++, Visual Basic, C#, etc.
- Generate EJB components, servlets, JSP, ASP.NET
- Generate a Conceptual Data Model (CDM),a Physical Data Model (PDM) and an Object-Oriented Model from the OOM
- Reverse engineer Java, XML files, PowerBuilder objects
- Import a Conceptual Data Model (CDM)
- Import a Physical Data Model (PDM)
- Import a Rose model
- Import/export XMI
- Customize the OOM to suit physical and performance considerations
- Customize and print model reports

UML and object-oriented modeling

What is UML?

UML (Unified Modeling Language) is a modeling language aimed at defining standards for object-oriented modeling. UML has become a standardized language largely through the work of the OMG (Object Management Group), a group composed of individuals and representatives of companies involved in object-oriented projects. However, its original conception drew much of its inspiration from the work of Grady Booch, James Rumbaugh, and Ivar Jacobson.

UML has a vocabulary and rules that focus on the conceptual and physical representation of a system. You use UML symbols and notations to create your models and diagrams.

Notational Terminology

UML has a well-defined syntax and semantics that is clear and easy to use in object modeling. The terminology used in the OOM interface is consistent with UML language notations.

What is object-oriented modeling?

Object-oriented modeling refers to the process of using objects as the basic building blocks for creating a software system. An object in this context usually means a class, that is, a description of a set of common objects. Each object or class has identity and behavior. They communicate with each other through different types of relationships which have their own symbols.

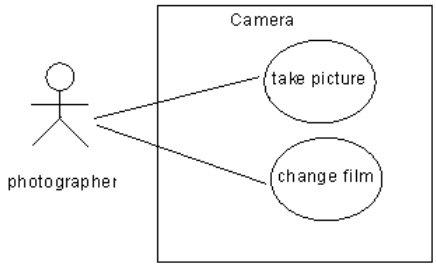
You use these objects to build models in which the properties of each object interact to perform certain actions that together make up a system of information.

For more information on object modeling, a bibliography is proposed in the About this book section of this manual.

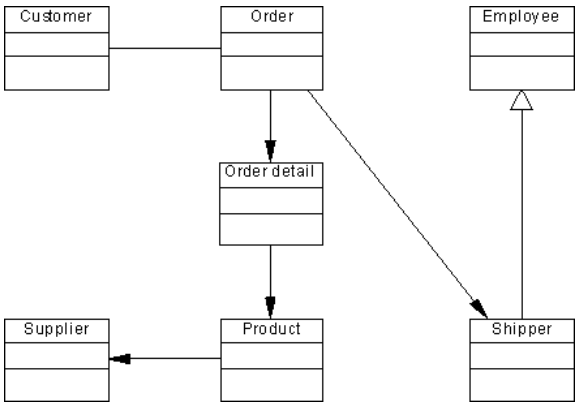
What is an OOM?

You use an Object-Oriented Model to build an OOM. An OOM is a structure which provides a close description of a system using the following UML diagrams:

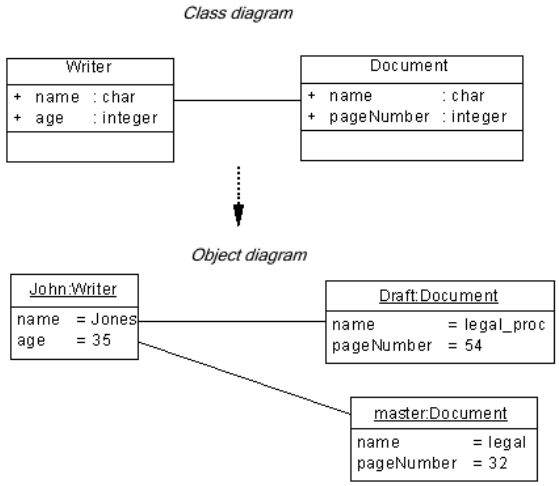
- **Use case diagrams** define the fundamental structure of your application. It allows you to draw a high-level sketch of the system: you define who are the external actors of the system and what activities they perform using it. The following example shows a use case of an actor Photographer performing two things: take pictures and change the film. Obviously, when he takes a picture, he has to switch the flash on, open the shutter, and then close the shutter but these activities are not of a high enough level to be represented in a use case.



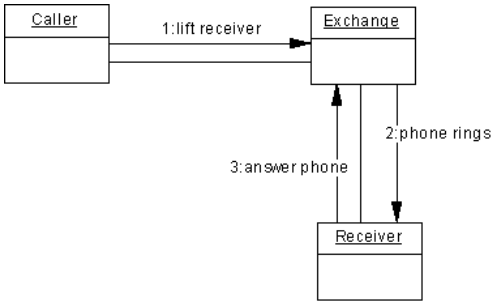
- **Class diagrams** define the static structure of the model. The class diagram contains a set of packages, classes, interfaces, and their relationships. These objects together form a class structure that is the logical design view of all, or part of a software system. The following example shows the process of product order and shipment in a class diagram.



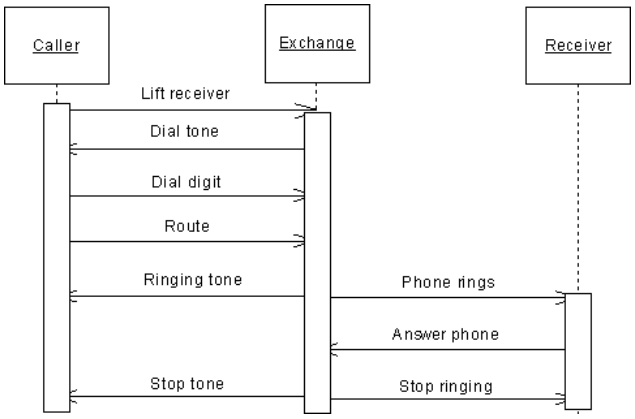
- **Object diagrams** describe the structure of model elements. They complement class diagrams, and display objects (instances of class), instance links (instances of association), and dependencies. The class diagram describes classes with associations and dependencies between them, and the object diagram goes deeper to highlight the constraints between objects of the classes. The following example is an object diagram derived from of a class diagram:



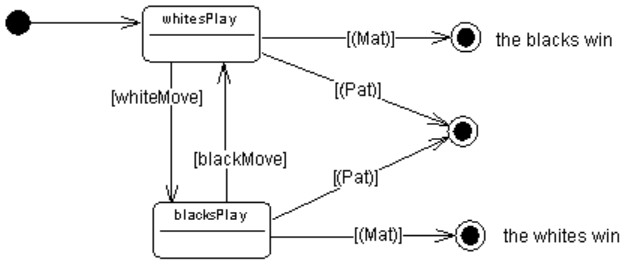
- **Collaboration diagrams** describe how the system achieves what was described in a use case diagram. They convey the same information as sequence diagrams, the main difference is that collaboration diagrams focus on objects in action, they show a network of objects that are collaborating.



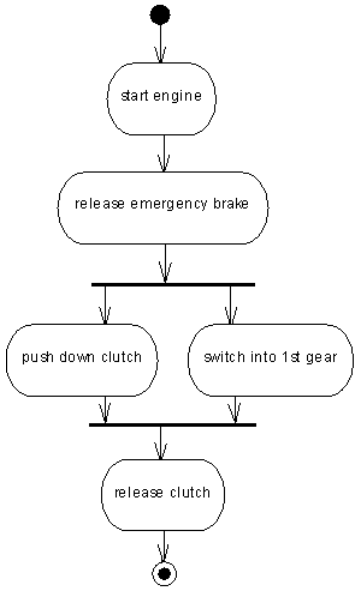
- **Sequence diagrams** describe how the system achieves what was described in a use case diagram. You use a sequence diagram to draw objects (instances of a class), and show a time-ordered series of sequenced method invocations among objects. The following example shows the mechanism of a phone call in a sequence diagram.



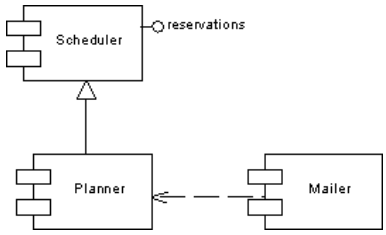
- **Statechart diagrams** describe the public behavior of a unique classifier (use case, component or class). Statechart diagrams show a finite number of states that the classifier can have, and the transitions that cause a change in state. The following example shows the possible states and exchanges between them:



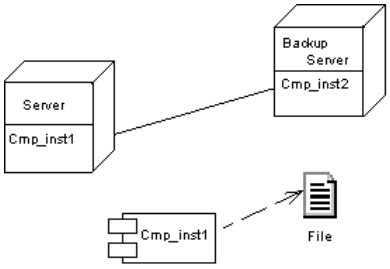
- **Activity diagrams** model the dynamic aspects of a system. They describe the flows driven by internal processing from a start point to several potential end points. They represent the internal behavior of a model element: use case, package, classifier or operation. The following example of activity diagram shows the flow of activities when starting a car.



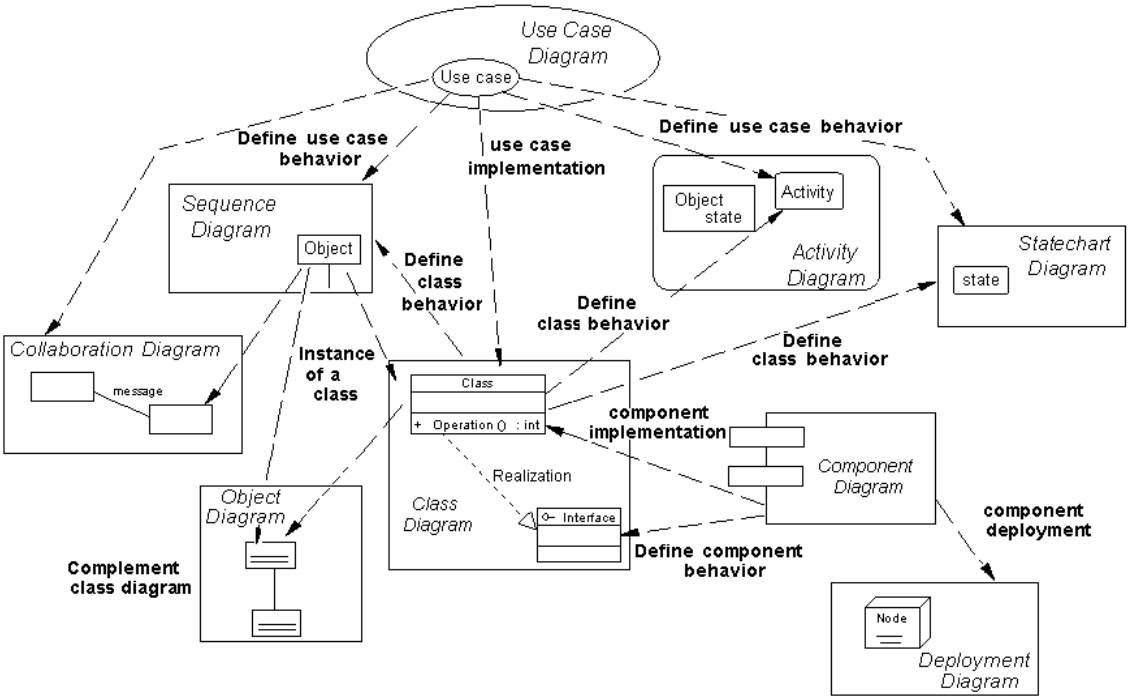
- **Component diagrams** show the dependencies among software components, including source code components, binary code components, and executable components. The following example shows relationships between components in a showroom reservation system:



- **Deployment diagrams** are implementation diagrams that complement component diagrams by giving more accurate details about the physical implementation and interactions of components.



In the picture below, you can see how use case, class, object, collaboration, sequence, statechart, activity, component, and deployment diagrams can interact within the same model:



Working with diagrams in an OOM

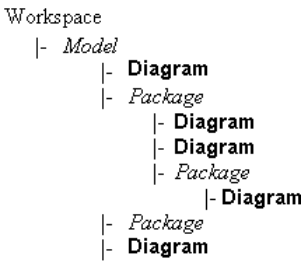
You can create as many use case, class, object, collaboration, sequence, statechart, activity, component, and deployment diagrams as you need in the same Object-Oriented Model (OOM).

All diagrams have a name and graphical contents. They are projections of the model and they represent it under different angles. They are sorted alphabetically in the Browser, except the default diagram that is the first in the list.

Moreover, each diagram has its own icon to help you quickly identify its type in the Browser:

Diagram type	Icon	View
Use case diagram		Static view
Class diagram		Static view
Object diagram		Static view
Collaboration diagram		Dynamic view
Sequence diagram		Dynamic view
Statechart diagram		Dynamic view
Activity diagram		Dynamic view
Component diagram		Static implementation view
Deployment diagram		Static deployment view

Packages



You can create any type of diagrams in a package, or in a model. You use packages to organize your objects into functional or organizational domains.

If you delete a package, you also delete the diagrams and the objects it contains. Objects created in a model or a package are shared between the diagrams of same type created in the same model or package.

For more information on packages, see section Defining packages in an OOM.

Objects in an OOM

Objects belong to models and their symbols are displayed in diagrams.

You can create any object from the Browser, it is not dependent on the active diagram, but if you create objects using the tool palette, the objects available are specific to the active diagram.

Use case diagram

Object	Tool	Description
Package		Used to organize elements into groups
Actor		Used to represent an external person, process or something interacting with a system, sub-system or class
Use case		Defines a piece of coherent behavior in a system, without revealing its internal structure
Association		Communication path between an actor and a use case that it participates in
Generalization		A link between a general use case and a more specific use case that inherits from it and add features to it
Dependency		Relationship between two modeling elements, in which a change to one element will affect the other element

Class diagram

Object	Tool	Description
Package		Used to organize elements into groups
Class		Set of objects sharing the same attributes, operations, methods, and relationships
Interface		Descriptor for the externally visible operations of a class, object, or other entity without specification of internal structure
Attribute	—	Named property of a class
Operation	—	Service that can be requested from a class
Association		Structural relationship between objects of different classes
Inner link		Exists when a class is declared within another class or interface
Aggregation		A form of association that specifies a part-whole relationship between a class and an aggregate class (example: a car has an engine and wheels)
Composition		A form of aggregation but with strong ownership and coincident lifetime of parts by the whole; the parts live and die with the whole (example: an invoice and its invoice line)
Dependency		Relationship between two modeling elements, in which a change to one element will affect the other element
Realization		Semantic relationship between classifiers, in which one classifier specifies a contract that another classifier guarantees to carry out.
Generalization		Link between classes showing that the sub-class shares the structure or behavior defined in one or more superclasses

Object diagram

Object	Tool	Description
Package		Used to organize elements into groups
Object		Instance of a class
Attribute values	—	An attribute value represents an instance of a class attribute, this attribute being in the class related to the object
Instance link		Communication link between two objects
Dependency		Relationship between two modeling elements, in which a change to one element will affect the other element

Collaboration diagram

Object	Tool	Description
Package		Used to organize elements into groups
Actor		Used to represent an external person, process or something interacting with a system, sub-system or class
Object		Instance of a class
Instance link		Communication link between two objects
Message		Communication that conveys information with the expectation that action will ensue. It creates an instance link by default when no one exists

Sequence diagram

Object	Tool	Description
Package		Used to organize elements into groups
Actor		Represents an external person, process or something interacting with a system, sub-system or class
Object		Instance of a class
Activation		Execution of a procedure, including the time it waits for nested procedures to execute

Object	Tool	Description
Message		Communication that conveys information with the expectation that action will ensue
Self Message		Recursive message: the sender and the receiver is the same object
Call Message		Procedure call message with a default activation
Self Call Message		Procedure call recursive message with a default activation
Return Message		Generally associated with a Procedure Call, the Return message may be omitted as it is implicit at the end of an activation
Self Return Message		Recursive message with a Return control flow type

Statechart diagram

Object	Tool	Description
Package		Used to organize elements into groups
Start		Starting point of the states represented in the statechart diagram
State		Defined as the situation of a model element waiting for events
Action	—	Specification of a computable statement
Event	—	Occurrence of something observable, it conveys information specified by parameters
Transition		Path the control flow transits on to join states
Junction point		Similar to the decision in the activity diagram except that it accepts several input and output transitions
End		Termination point of the states described in the statechart diagram

Activity diagram

Object	Tool	Description
Package		Used to organize elements into groups
Start		Starting point of the activities represented in the activity diagram
Activity		Invocation of an action
Composite activity	—	Complex activity decomposed to be further detailed
Object state		Represents a specific state of an activity
Organization unit		Element that can represent a company, a system, a service, an organization, a user or a role
Transition	→	Path the control flow transits on to join activities
Decision	◇	Decision the control flow has to take when several transition paths are possible
Synchronization		Enables synchronization of control between two or more concurrent actions
End		Termination point of the activities described in the activity diagram

Component diagram

Object	Tool	Description
Package		Used to organize elements into groups
Component		Represents a shareable piece of implementation of a system
Dependency		Relationship between two modeling elements, in which a change to one element will affect the other element
Generalization		A link between a general component and a more specific component that inherits from it and add features to it

Deployment diagram

Object	Tool	Description
Package		Used to organize elements into groups
Node		Physical element that represents a processing resource, a physical unit (computer, printer, or other hardware units)
Component instance		Instance of a deployable component that can run or execute on a node
Node association		An association between two nodes means that the nodes communicate to each other
Dependency		Relationship between two modeling elements, in which a change to one element will affect the other element

Defining an OOM

You can create a new OOM, open an existing one, close or detach an OOM from the workspace. An OOM is attached to one object language that must be selected at creation.

Object language overview

An object language contains specifications for a particular language. It provides PowerDesigner with the syntax and guidelines for implementing stereotypes, data types, scripts and constants for an object language.

Each OOM is by default attached to an object language. When you create a new OOM, you must choose an object language. You can create a new object language or use the languages delivered with PowerDesigner.

The definition of an object language is available from its property sheet. You can select and configure parameters used when defining objects or generating from an OOM.

For more information on object languages, see chapter Object Languages Reference Guide in the *Advanced User Documentation*.

In an OOM, you can use the List of Object languages to display the properties of object languages, and to compare, or merge object languages.

For more information on using the list of object languages, comparing or merging object languages, see chapter The Resource Editor in the *General Features Guide*.

Extended dependencies

The Extended Dependencies page in property sheets allows you to define your own semantic dependencies between objects of the model. They are not interpreted and checked by PowerDesigner. These dependencies can be created to document a special relationship between objects, they are meant to be used for documentation purposes only.

Each time an extended dependency is defined on the dependent object, the influent object displays the inverse relationship in the Extended Influences tabbed page of the Dependencies page.

Example

You could create an extended dependency between an actor and a file to attach a .RTF file to an actor. It will describe the actor in a .RTF format instead of using the Annotation tabbed page in the Notes page. In this case, you can use a <<documentation>> stereotype in the Extended Dependencies page of the actor property sheet.

For more information on Extended Dependencies, see section Using extended dependencies, in chapter Using the PowerDesigner Interface in the *General Features Guide.*

Creating an OOM

There are several ways to create an OOM:

- Create a new OOM
- Import one or more existing OOM
- Generate an OOM from a CDM
- Generate an OOM from PDM
- Import a Rational Rose model (.mdl)
- Reverse engineer an object language into an OOM

When you create a new OOM, you have to define creation parameters.

Object Language selection

In PowerDesigner, the current object language is a text file in XML format that contains all the syntax and specifications for a target language.

PowerDesigner has an object language file for each supported language. You can customize your object language. All object languages are stored in the Object Languages directory in your installation path..

For more information on customizing an object language, see section Understanding the object language editor, in chapter Object Languages Reference Guide in the *Advanced User Documentation*.

Each object language has the extension .XOL which signifies XML Object Language.

You have to attach the new OOM to a target language, for example Java or XML Schema. When you attach a language to a model, you can choose one of the following options:

Option	Definition
Share	Current object language constantly refers to the object language stored in the Resource Files\Object Languages directory. Any changes made to the object language are shared by all linked OOM
Copy	Current object language is a unique copy of the object language stored in the Resource Files\Object Languages directory. The current object language is independent of the original object language, so modifications made to the object language in the Object Languages directory are not available to the OOM. The copied object language is saved with the OOM and cannot be used without it

Diagram selection

You also have to select the type of the first diagram in the new model. The First Diagram dropdown listbox lets you select one of the following diagrams:

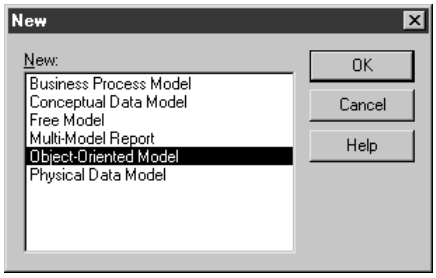
- Use Case diagram to specify a high-level sketch of the system with actors and what they do with the system
- Class diagram to define the static structure of the model with classes and interfaces, their internal structure and links
- Object diagram to complement the class diagram and define the structure of objects
- Collaboration diagram to describe how the system achieves what was described in the use case diagram using collaboration between objects of the model
- Sequence diagram to describe how the system achieves what was described in the use case diagram using a chronological scale
- Statechart diagram to describe a finite number of states that a classifier can have, and the events that generate transitions between states
- Activity diagram to describe a low-level, detailed view of a system represented as a workflow between activities
- Component diagram to show dependencies among software components
- Deployment diagram to complement component diagrams by giving more details about the physical implementation and interactions of components

The first type of diagram selected remains in memory: the next time you create a new model, the same type of diagram appears in the First Diagram dropdown listbox.

❖ To create a new OOM:

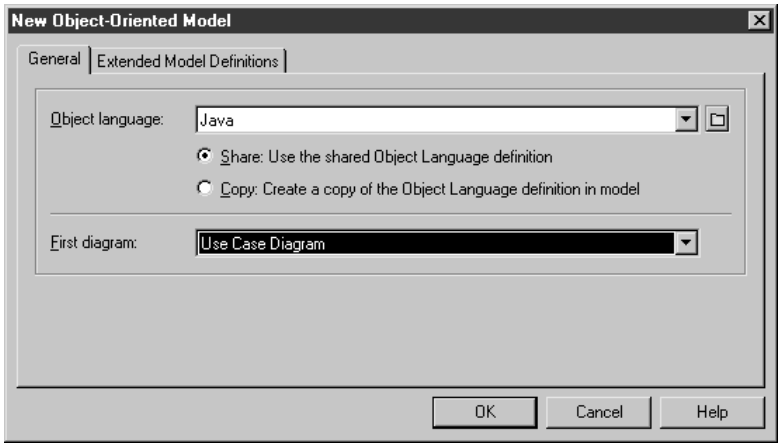
1 Select File→New.
or
Click the New button in the toolbar.

The New window appears.



2 Select Object-Oriented Model and click OK.

The New Object-Oriented Model dialog box appears.



3 Select an object language from the Object language dropdown listbox.

For more information on object language properties, see section Understanding the object language editor, in chapter Object Languages Reference Guide in the *Advanced User Documentation*.

4 Select either Share or Copy.

5 Select the type of the first diagram in the First Diagram dropdown listbox.

6 <optional> If you want to attach one or more extended model definitions to complement the selected object language, click the Extended Model Definitions tab, and select the extended model definitions of your choice.

For more information on attaching extended model definition to a model, see section Selecting extended model definitions at model creation.

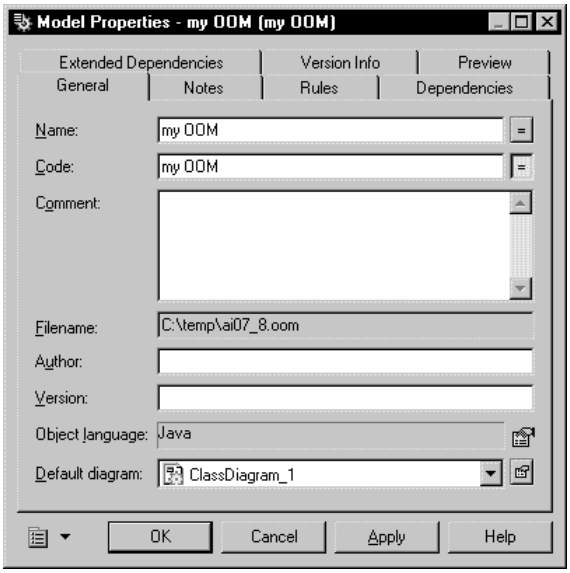
7 Click OK.

A default workspace is visible in the Browser tree view, the new OOM is created in this workspace.

8 Select Model→Model Properties.
or
Right-click any empty space in the diagram window and select Properties from the contextual menu.

The model property sheet appears.

9 Type a model name and code.



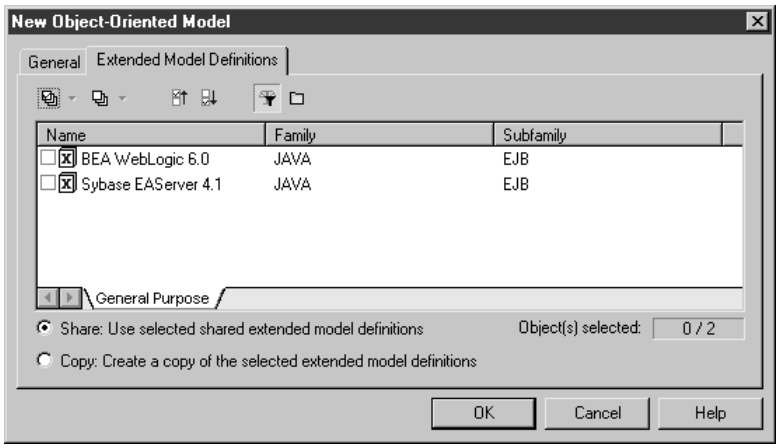
10 Click OK.

Demo example
An example of an OOM is available in the Examples directory.

Selecting extended model definitions at model creation

Extended model definitions (.XEM files) provide means for customizing and extending PowerDesigner metaclasses, parameters and generation. Extended model definitions are typed like models in PowerDesigner. You create an extended model definition for a specific type of model and you cannot share these files between heterogeneous models.

When you create a new OOM, or when you reverse into a new OOM, you can select one or several extended model definitions and attach them to the model from the New Object-Oriented Model dialog box:



When you create an OOM in Java or .NET languages, the Extended Model Definitions page in the New Object-Oriented Model dialog box contains the WSDL extended model definition file. It is attached by default to these object languages at creation of the model.

When you attach an extended model definition to an object language, you can choose one of the following options:

Option	Definition
Share	Current extended model definition constantly refers to the extended model definition stored in the Resource Files\Extended Model Definitions directory. Any changes made to the extended model definition are shared by all linked XEM
Copy	Current extended model definition is a unique copy of the extended model definition stored in the Resource Files\Extended Model Definitions directory. The current extended model definition is independent of the original one, so modifications made to the extended model definition in the Resource Files\Extended Model Definitions directory are not available to the copied XEM. This one is saved with the OOM and cannot be used without it

For more information on extended model definitions, see chapter Extended Model Definitions Reference Guide, in the *Advanced User Documentation.*

Changing the object language of an OOM

You can change the object language of a model. If you change the object language of a model, you have to define the status of the language:

Object language	Description
Share	Use the shared object language stored in the Object Languages directory of your installation. Any changes made to the object language are available to the linked OOM
Copy	Create a copy of the object language in the model. The current object language is independent from the original object language so any changes made in the object language are not available to other models. The object language is saved with the model and cannot be used by other models

> ***Caution***
> *PowerDesigner is delivered with a set of object languages. It is strongly advised to make a backup copy of each object language before you start modifying them.*

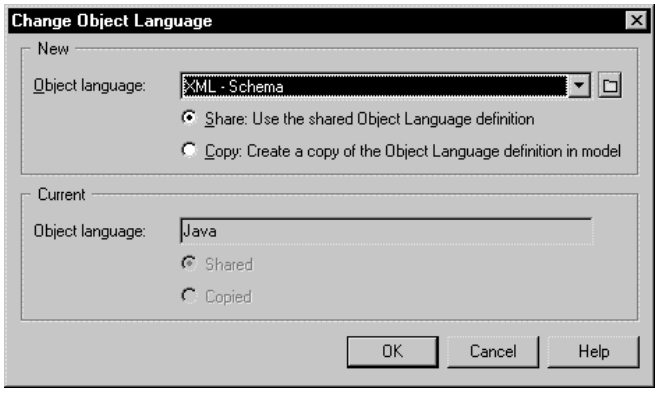
❖ **To change the object language of a model:**

1 Select Language→Change Current Object Language.

The Change Object Language dialog box appears.

2 Select an object language.

3 Select the Share or Copy radio button.



4 Click OK.

A message box informs you that the current object language has been changed.

5 Click OK.

Opening an existing OOM

An OOM has the file extension .OOM.

Choose object language

If PowerDesigner cannot find the object language attached to the Object-Oriented Model you want to open, the Choose Object Language dialog box appears to let you select another object language to attach to the model. You can select the object language from the Object language dropdown listbox.

❖ **To open an existing OOM:**

1 Select File→Open.
or
Click the Open tool.

A standard Windows file selection dialog box appears.

2 Select a file with the .OOM extension.

3 Click Open.

The model opens in the diagram window and appears in the Browser.

Replacing an object language from previous version
When you open a model linked with an object language from a previous version, PowerDesigner automatically displays a confirmation dialog box to let you replace the object language of your model. Object languages have been modified in the current version and it is highly recommended to accept this change otherwise you will not be able to generate for the selected language.
For more information on how to replace object languages, see section Replacing an object language from version 7 or 8, in document Migrating from previous versions, available in the PowerDesigner Printable Docs directory.

Detaching an OOM from the Workspace

You can detach an OOM from a workspace. When an OOM is detached from a workspace, its node is removed from the Browser and it is no longer defined in the workspace, but the file is not deleted from your operating environment.

❖ **To detach an OOM from the Workspace:**

1 Right-click the OOM node in the Browser and select Detach From Workspace from the contextual menu.

A confirmation box asks if you want to save the OOM.

2 Click Yes if you want to save modifications to the OOM.
Select or browse to a directory.
Type a name for the file and click the Save button.
or
Click No if you don't want to save modifications to the file.

The OOM is removed from the Workspace.

Defining OOM properties

The Model property sheet displays the definition of the current model.

An OOM has the following model properties:

Property	Description	Length
Name	Name for the model	254
Code	Code for the model. This code is generated in database scripts	254
Comment	Descriptive label for the model	__
Filename	Location of the model file. This box is empty if the model has never been saved	__
Author	Author of the model. You can insert a name, a space, or nothing. If you insert a space, the Author field in the title box remains empty. If you intentionally leave the box empty, the Author field in the title box displays the user name from the Version Info page of the model property sheet	254
Version	Version of the model. You can use this box to display the repository version or a user defined version of the model. This parameter is defined in the display preferences of the Title node	__
Object language	Current object language for the model	—
Default diagram	Diagram displayed by default when opening the model	—

❖ To define the properties of an OOM:

1 Select Model→Model Properties.
or
Right click the diagram background and select Properties from the contextual menu.

The model property sheet appears.

2 Type changes to model properties.

If you want to change the object language, click the Properties tool to the right of the box to display the property sheet of the object language.

3 Click OK.

Previewing the code of an OOM

You can preview the code of a model or package in order to see the generated code, by clicking the Preview tab. The displayed code changes according to the object language you have selected in your model. For example, if you have created EJB or servlet components in Java, the Preview page displays the EJB or Web deployment descriptor files. If you have selected an XML family language, the Preview page displays the Schema file that corresponds to the XML file to be generated.

You can use the following tools and accelerators from the Preview toolbar:

Tool	Description	Accelerator
	Open Editor Contextual menu	SHIFT + F11
	Refresh	F5
	Show Generation Options	CTRL + W
	Select Generation Targets	CTRL + F6

Showing generation options

If you click the Show Generation Options tool, the Generation Options dialog box appears. You can change generation options from this dialog box.

Select Generation Targets

When available, this tool displays the list of target extended model definitions selected for generation. If you add a generation target, the corresponding tab is added to the Preview page. If you deselect a generation target, the corresponding tab disappears from the Preview page. This tool is disabled if no target extended model definition is attached to the model.

For more information on the generation targets, see section Defining generation targets (XEM), in chapter Generating for a language.

Script bookmarks

In the Preview page, you can add and remove bookmarks at specific points in the code and navigate forwards or backwards from bookmark to bookmark:

Accelerator	Description
CTRL + F2	Adds a new bookmark. A blue bookmark box appears. If you repeat this action from the same position, the bookmark is deleted and the blue marker disappears
F2	Jumps to bookmark
SHIFT + F2	Jumps to previous bookmark

Note that bookmarks are not printable and are lost if you use the Refresh, or the Show Generation Options tools.

Modifying the code using Preview

You can modify the code directly from the Preview page of a classifier, it is a quick and easy way to apply modifications to your code and update the model. The code modified must contain a valid definition otherwise the changes are ignored. Your changes only apply to the current classifier, therefore you are not allowed to:

- Move the classifier to another package by changing the package declaration
- Rename the classifier using the name of another existing classifier

You are allowed to create generalization and realization links provided their classifiers already exist in the model, if the classifiers do not exist, then creation is ignored.

If you want to modify the code, you have to select the Preview Editable check box from Tools→Model options. This option is applicable to the entire model, and is available for the **Java** language.

When the Preview Editable check box is selected, the changes apply as soon as you leave the Preview page, or you click the Apply button. Whenever applied, your changes are also displayed in the Output window located in the lower part of the main window.

> ***Caution***
> *You should avoid renaming attributes and operations because all attributes that are not generated, like description, annotation or extended attributes will be lost.*

Defining OOM model options

You can set options to define how your model is created or modified, and the type of information that it displays. Those options apply to the model and to all its objects.

You can set model options on the following elements:

- All objects
- Default data types
- Domain/Attribute
- Message

You can also set naming conventions for each type of objects in your model.

For more information on naming conventions, see section Defining naming conventions, in chapter Managing Models in the *General Features Guide*.

All objects

You can set the following option for classes in an OOM:

Option	Description
Name/code case sensitive	From the All Objects groupbox, you can define the case sensitivity of names and codes in the current model. Unlike other model options, you can modify the name and code case sensitivity during the design process. However, if you do so, make sure you run the check model feature to verify if the model does not contain any duplicate object
Show classes as data types	Includes classes of the model in the list of data types defined for attributes or parameters, and return types defined for operations
Preview editable	Applies to reverse engineering. You can edit your code from the Preview page of a class by selecting the Preview Editable check box. This allows you to reverse engineer changes applied to your code directly from the Preview page

For more information on using the Preview page, see section Modifying the code using Preview.

Default data types

The default data type applies to attributes, operations and parameters if no data type is selected.

You can use the different dropdown listboxes to define a default data type for each new attribute, operation, or parameter.

Domain/Attribute

Enforce non-divergence

You can choose to enforce non-divergence between a domain and the attributes using the domain. When you select the Enforce non-divergence check box, all attributes attached to a domain must not diverge from the domain. You can also define which attribute property is not allowed to diverge from the domain.

Changes on domain non-divergence options apply only to the current OOM.

Property	Attributes in the domain cannot have divergent
Data type	Data type
Check	Check parameters
Rules	Business rules

Your choice of whether or not to enforce domain and attribute non-divergence has the following results:

Non-divergence	Result
Not enforced	Attributes that diverge from the domain can remain attached to the domain
Enforced	Attributes that diverge from the domain (for certain attribute properties) must be detached from the domain

Your selection to enforce, or not to enforce non-divergence between a domain and the attributes using the domain, has the following results on the attribute properties (data type, check, or rules):

Non-divergence	Property is…	Result when domain is modified
Enforced	Selected	The property is updated for all attributes using the domain
	Not selected	No property is updated
Not enforced	Selected	You must confirm property update for all attributes using the domain
	Not selected	No property is updated

When the Enforce non-divergence check box is not selected, if you select and modify one, or more attribute property so that it diverges from its current domain, you are asked if you want to keep the modifications and diverge from the domain. If you do not select any attribute property and you modify an attribute or a domain property, you create a divergence; in this case you are not warned, and you can no longer impact the change on the attribute properties.

If you modify any of the Minimum, Maximum, and Default check parameters of a domain used by an attribute, only the modified domain check parameter is copied to the attribute, not all attribute check parameters are replaced by domain check parameters.

If you modify the constraint of the Minimum, Maximum, and Default check parameters of either domain or attribute properties, the attribute properties must always enforce the constraint defined for the domain.

For example, Domain_1 has the following check parameters: Minimum = 5, Maximum = 15, and Default = 10. Attribute_1 uses Domain 1. If you change the minimum check parameter of Domain_1 to 3, this value is not replaced in the attribute properties when you click Apply. This is because 5 respects the constraints defined by the domain.

Use data type full name

When selected, uses the full data type name instead of its abbreviated from. It applies to attribute data types, and provides a clear persistent data type list for attributes.

Message

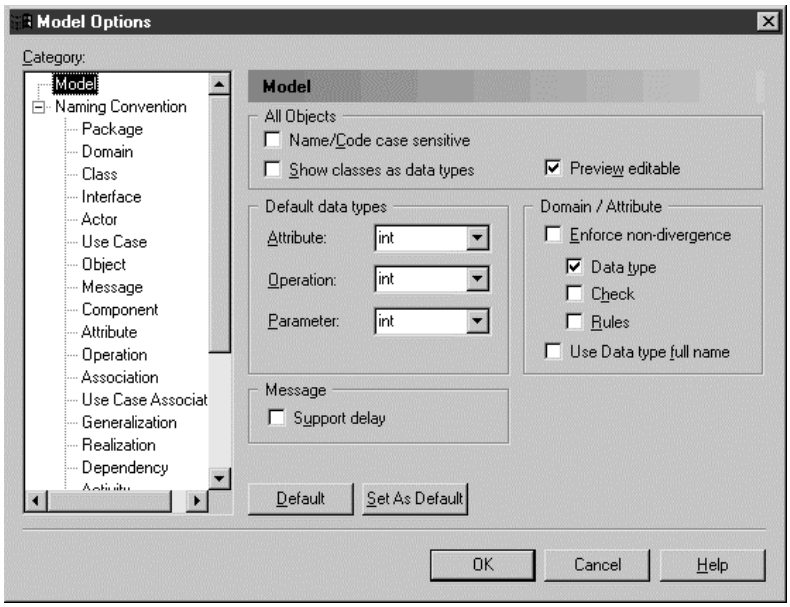
You can set the following option for messages:

Option	Description
Support delay	If selected, implies that messages may have duration (slanted arrow message) If deselected, implies that messages are instantaneous, or fast (horizontal message)

❖ **To define OOM model options:**

1 Select Tools→Model Options.
or
Right-click any empty space in the diagram and select Model Options from the contextual menu.

The Model Options dialog box appears.



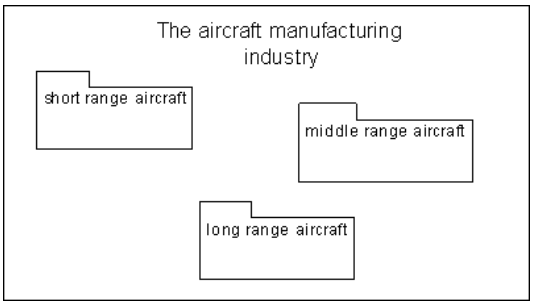
2 Select model options in the different boxes.

3 Click OK.

Defining packages in an OOM

A **package** is a general purpose mechanism for organizing elements into groups. It contains model objects and is available for creation in all diagrams.

When you work with large models, you can split them into smaller subdivisions to avoid manipulating the entire set of data of the model. Packages can be useful to assign portions of a model, representing different tasks and subject areas to different development teams.



Package hierarchy

You can create several packages at the same hierarchical level within a model or decompose a package into other packages and continue this process without limitation in decomposition depth. Each package appears with a default diagram window. At each level of decomposition you can create several diagrams.

However, in an activity or statechart diagram, it is not possible to create a package in a composite activity or composite state since those ones are behaving like a package.

For more information on packages, see section Defining a package, in chapter Managing Models in the *General Features Guide*.

OOM Package properties

Packages have properties displayed on property sheets. All packages share the following common properties:

Property	Description	Length
Name	Name that clearly identifies the package	254
Code	Codes are references for packages	254
Comment	Optional label that describes a package and provides additional information	—

Property	Description	Length
Stereotype	Sub-classification derived from an existing package	254
Use parent namespace	Defines the package as being the area in which the name of an object must be unique in order to be used	—
Default diagram	Diagram displayed by default when opening the package	—

Stereotype

A package has the following default stereotypes:

Stereotype	Description
<<facade>>	Package is a view of another package
<<framework>>	Package consists mostly of patterns
<<jar>>	Jar archive
<<metamodel>>	Package is an abstraction of another package
<<model>>	Specifies a semantically closed abstraction of a system
<<stub>>	Package serves as a proxy for the public contents of another package
<<subsystem>>	Grouping of elements, some of which constitute a specification of the behavior offered by the other contained elements
<<system>>	Package represents the entire system being modeled
<<systemModel>>	Package that contains other packages with the same physical system. It also contains all relationships and constraints between model elements contained in different models
<<topLevel>>	Indicates the top-most package in a containment hierarchy

Previewing the code of a package in an OOM

You can preview the code of a model or package in order to see the generated code, by clicking the Preview tab. The displayed code changes according to the object language you have selected in your model.

For more information on the code preview, see section Previewing the code of an OOM

Defining the diagram type of a new package

When you create a new package, the default diagram of the package is defined according to the following parameters:

- If you create a package using the Package tool from the palette, the diagram is of the same type as the parent package or model.
- If you create a package from the Browser, the diagram is of the same type as existing diagrams in the parent package or model, if these diagrams share the same type. If diagrams in the parent package or model are of different types, you are asked to select the type of diagram for the new sub-package.
- If you create a package from the List of Packages, the diagram is of the same type as the active diagram.

Modifying package display preferences

You can modify the following display preference for packages using the Tools→Display Preferences command:

Preference	Description
Show stereotype	Displays the stereotype of the package

CHAPTER 2

Building a Use Case Diagram

About this chapter

This chapter describes how to build a use case diagram. It explains the role of each element in a use case diagram and how to create and modify them.

Contents

Topic	Page
Use Case basics	36
Defining use cases	38
Defining actors in a use case diagram	44
Defining associations in a use case diagram	50
Defining generalizations in a use case diagram	54
Defining dependencies in a use case diagram	59

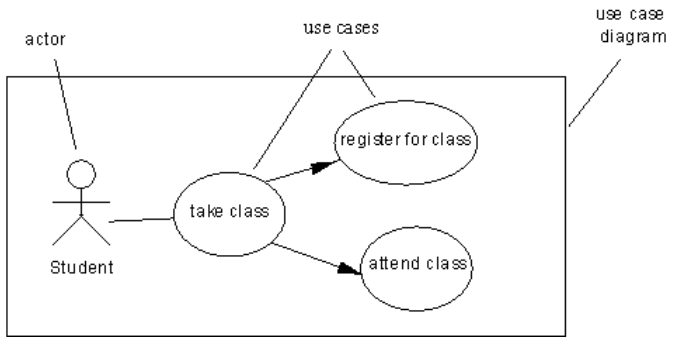
Use Case basics

A **use case diagram** is a diagram used to define the behavior of a system. It shows interactions between actors (the external users) and use cases (the services performed in the system).

Defining a use case diagram

The use case diagram is one of the diagrams of the Unified Modeling Language (UML). It is a high level model for requirements and system behavior design. This diagram graphically models the functionality of the system as perceived by outside users, called actors.

The use case diagram is part of an Object-Oriented Model (OOM). It is one of the available diagrams in an OOM. It shows the symbols of the objects defined in the system you are describing. The objects are the actors, use cases, and the relationships among them. The use case diagram displays which actor participates in which use case.



With a use case diagram, you immediately see a snapshot of the system's functionality. Further details can later be added to the diagram if you need to elucidate interesting points in the system's behavior.

Why build a use case diagram?

A use case diagram describes what a system will do, with a user focus in order to envision a project and give the application some structure. A use case can be later defined into other use cases if you need to further describe how a global service is planned and delivered.

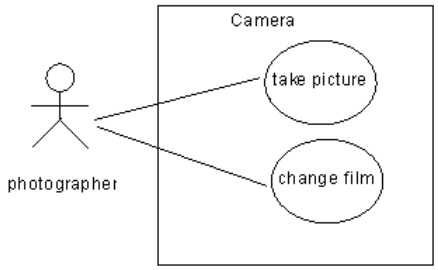
Applicability

A use case diagram is well suited to the task of describing all of the things that can be done with a database system by all the people who might use it.

Inversely, a use case diagram is poorly suited to describing the TCP/IP network protocol because there are many exception cases, branching behaviors, and conditional functionality (what happens when the connection dies, what happens when a packet is lost?)

Example

The following example shows the use case of a camera, the actor photographer performs two things with his camera: take pictures and change the film. Obviously, when he takes a picture, he has to switch the flash on, open the shutter, and then close the shutter but these activities are not of a high enough level to be represented in a use case.



For more information on diagrams of the OOM, see section What is an OOM, in chapter Object-Oriented Model Basics.

Creating a use case diagram

You can create a use case diagram using different methods.

For more information on creating a use case diagram in a new OOM, see section Creating an OOM, in chapter Object-Oriented Model Basics.

For more information on creating a use case diagram in an existing OOM, see section Creating a new diagram, in chapter Managing Models, in the *General Features Guide*.

Defining use cases

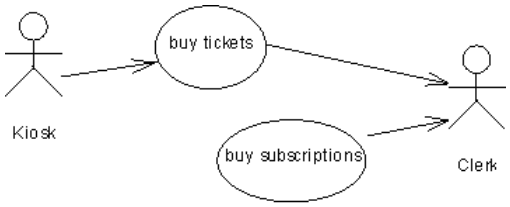
A **use case** could be defined as a service that a system performs.

A use case defines a piece of behavior of a classifier (a system, a sub-system, or a class) as it appears to an external user, without revealing the internal structure of the classifier.

Its purpose is to specify a sequence of actions that a system, or sub-system can perform, interacting with actors of the system.

Example

In our example, buy tickets and buy subscriptions are use cases.



Use case properties

You can double-click any use case symbol in a diagram to display its property sheet.

A use case has the following general properties:

Property	Description	Maximum length
Name	Name of the use case	254
Code	Reference name or code for the use case	254
Comment	Descriptive comment for the use case	—
Stereotype	Extends the semantics of a use case derived from existing use cases but specific to your needs	254

Use case specification

The Specification page contains the following tabbed pages:

Items	Description
Action Steps	Contains a text description of the normal sequence of actions associated with a use case Example: open a file, give a new registration number and write down medical treatment would be action steps for a use case called 'register patient' in a hospital
Extension Points	Contains a text description of actions that extend the normal sequence of actions. It opens up the use case to the possibility of extension (extend). Usually introduced with an if ….then Example: if the patient already has a registration number, then retrieve his personal file
Exceptions	Signal raised in response to behavioral faults during system execution
Pre-Conditions	A constraint that must be true when an operation is invoked
Post-Conditions	A constraint that must be true at the completion of an operation

The tabbed pages available from the Specification page contain a text zone. Their purpose is to help you give a more in-depth definition of a use case. The text can be saved in RTF or TXT format.

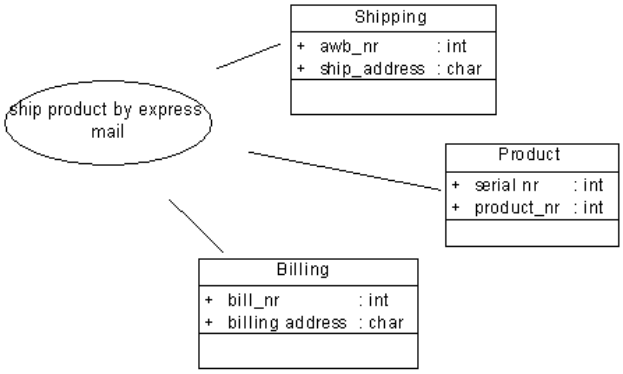
Accessing the Specification page
You can access the Specification page of a use case by right clicking the use case symbol in the diagram, and selecting Specification from the contextual menu.

Use case implementation classes

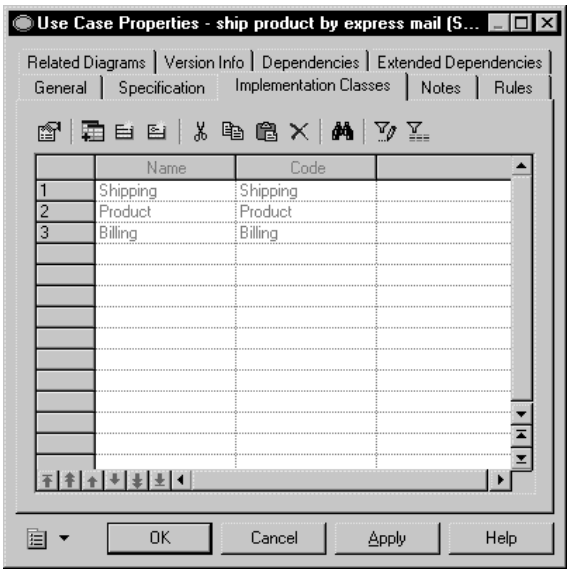
The Implementation Classes page shows the list of classes and interfaces used to implement a use case. A use case is a task or service, generally represented as a verb. When analyzing what a use case must do, it is necessary to identify the classes and interfaces that need to be created to fulfill the task.

In this logic, it is possible to create the classes or interfaces and attach them to a use case.

For example, a use case **ship product by express mail** needs the classes **Shipping**, **Product**, and **Billing** to perform its task. They are shown as related to the use case in the drawing below.



The implementation classes are the following:



Creating a class or an interface

You can create a class or interface from the Implementation Classes page using the Create a new class tool, or Create a new interface tool. It creates the class, or interface in the same package as the use case.

Tool	Action
	The new class is automatically added to the current list of implementation classes
	The new interface is automatically added to the current list of implementation classes

When the class or interface is defined in another package, a shortcut of the class or interface is created in the use case package.

For more information on implementation classes, see section Linking a class or an interface to a use case, in chapter Working with Object-Oriented Models.

Use case related diagrams

In the property sheet of a use case, you can select the Related Diagrams tab to display all diagrams of the model that are related to a use case. They can be attached and detached from the current use case at this stage. This page displays a list of the diagrams identified by their name and code.

For more information on Related Diagrams, see section Defining related diagrams, in chapter Working with Object-Oriented models.

Creating a use case

There are several ways to create a use case:

- From a diagram
- From the Browser
- From the List of Use Cases

At creation, a use case has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a use case, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a use case from a diagram:**

1 Click the Use case tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



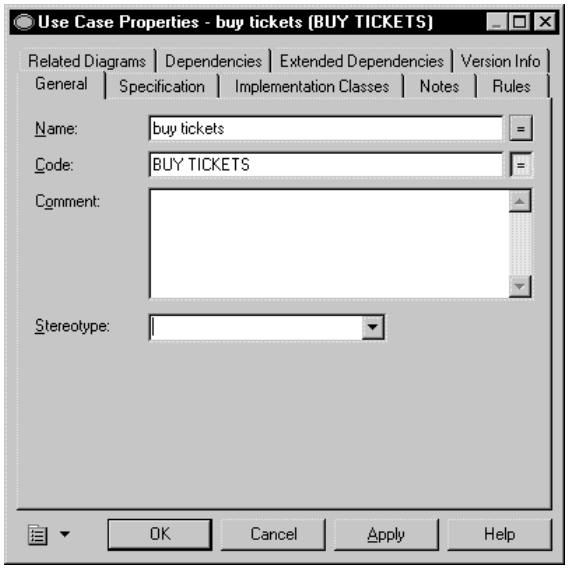
3 Click the Pointer tool.
or
Click the right mouse button.

You release the Use case tool.

4 Double-click the use case symbol to display its property sheet.

> **Opening property sheets at last accessed page**
> Property sheets open to the General page by default. However, you can choose to open property sheets at the last page accessed by selecting Tools→General Options→Dialog, and selecting the option Keep Last Tab in the Property Sheets groupbox.

5 Type a name and a code.



6 Click OK.

The use case node appears in the Use Cases category of the Browser.

Modifying use case properties

You can modify the use case properties using the following methods:

- From the use case property sheet
- From the List of Use Cases

For more information on the different ways to modify the use case properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying use case display preferences

You can modify the following display preferences for use cases using the Tools→Display Preferences command:

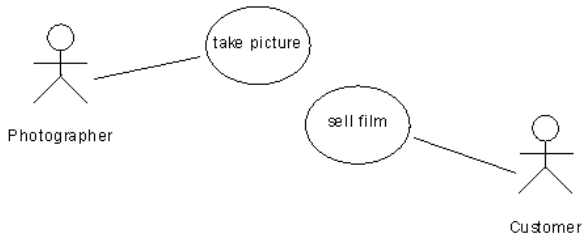
Preference	Description
Show stereotype	Displays the stereotype of the use case

Defining actors in a use case diagram

The system you are describing interacts with **actors**. An actor characterizes an outside user, or related set of users that interact with a system.

The most obvious candidates for actors are the humans in the system, but it can also be a system by itself. For example, actors in a computer network system may include a system administrator, a database administrator and users. Actors are typically those entities whose behavior cannot control, or change because they are not part of the system that you are describing.

Example



In the use case diagram, an actor can be a **primary actor** for a use case if it triggers the actions performed by a use case; the primary actor is the one who asks for an action to be performed by the use case. Primary actors are located on the left of the use case. To indicate that an actor is a primary actor for a use case, the association should be drawn from the actor to the use case.

An actor can be a **secondary actor** for a use case if it assists the use case to complete the actions but does not trigger the actions. After performing an action, the use case gives results, documents, or information to the outside: the secondary actor is the one who receives them. Secondary actors are located on the right of the use case. To indicate that an actor is a secondary actor for a use case, the association should be drawn from the use case to the actor.

On global scale, a secondary actor can also be a primary actor for another use case, in the same diagram or in another diagram.

An actor can be shared between a use case, a sequence, and a collaboration diagram if it plays the same role. Actors are common to the model. They can either be created in the diagram type you need, or dragged from a diagram type and dropped into another diagram type.

For more information on actors in the sequence diagram, see section Defining actors in a sequence diagram, in chapter Building a sequence diagram.

For more information on actors in the collaboration diagram, see section Defining actors in a collaboration diagram, in chapter Building a collaboration diagram.

Actor properties in a use case diagram

You can double-click any actor symbol in a diagram to display its property sheet.

An actor has the following general properties:

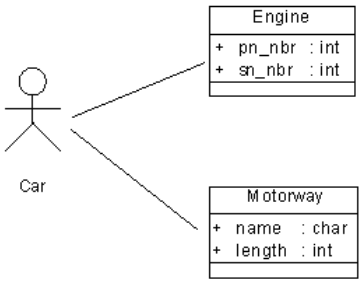
Property	Description	Maximum length
Name	Name of the actor	254
Code	Reference name or code for the actor	254
Comment	Descriptive comment for the actor	—
Stereotype	Extends the semantics of an actor derived from existing actors but specific to your needs	254

Actor implementation classes

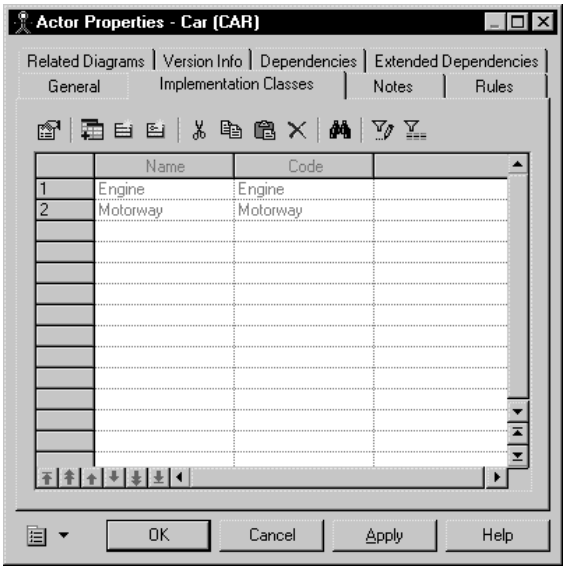
The Implementation Classes page shows the list of classes and interfaces used to implement an actor. An actor can be a human being (person, partner) or a machine, or process (automated system). When analyzing what an actor must do, it is necessary to identify the classes and interfaces that need to be created for the actor to perform his task.

In this logic, it is possible to create the classes or interfaces and attach them to an actor.

For example, an actor Car needs the classes Engine and Motorway to perform its task. They are shown as related to the actor in the following figure:



The implementation classes are the following:



Creating a class or an interface

You can create a class or interface from the Implementation Classes page using the Create a new class tool, or Create a new interface tool. It creates the class, or interface in the same package as the actor.

Tool	Action
	The new class is automatically added to the current list of implementation classes
	The new interface is automatically added to the current list of implementation classes

When the class or interface is defined in another package, a shortcut of the class or interface will be created in the actor package.

For more information on implementation classes, see section Linking a class or an interface to an actor, in chapter Working with Object-Oriented Models.

Actor related diagrams

In the actor property sheet, you can select the Related Diagrams tab to display all diagrams of the model that are related to an actor. They can be attached and detached from the current actor at this stage. This page displays a list of the diagrams identified by their name and code.

For more information on Related Diagrams, see section Defining related diagrams, in chapter Working with Object-Oriented models.

Creating an actor in a use case diagram

There are several ways to create an actor:

- From a diagram
- From the Browser
- From the List of Actors

At creation, an actor has a default name including a number, this number is assigned in the order of creation.

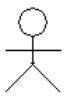
For more information on the different ways to create an actor, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an actor from a use case diagram:**

1 Click the Actor tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



3 Click the Pointer tool.
or
Click the right mouse button.

You release the Actor tool.

4 Double-click the actor symbol to display its property sheet.

5 Type a name and a code.



6 Click OK.

The actor node appears in the Actors category of the Browser.

Showing the symbol of an actor

You can show the symbol of an actor already created in a collaboration, or sequence diagram using the following methods:

- Drag and drop the symbol of an actor
- Select the symbol of an actor to display

Dragging and dropping the symbol of an actor

❖ **To drag an drop the symbol of an actor:**

1 Select the actor you need in the Browser.

2 Drag it and drop it into the use case diagram.

The actor symbol is displayed in the use case diagram.

Select the symbol of an actor to display

❖ **To select the symbol of an actor to display:**

1 Select Symbols→Show Symbols from the use case diagram.

The Show Symbols dialog box appears.

2 Select the actor to display.

3 Click OK.

The actor symbol is displayed in the use case diagram.

Modifying actor properties in a use case diagram

You can modify the actor properties using the following methods:

- From the actor property sheet
- From the List of Actors

For more information on the different ways to modify the actor properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying actor display preferences

You can modify the following display preferences for actors using the Tools→Display Preferences command.

Preference	Description
Show stereotype	Displays the stereotype of the actor

Defining associations in a use case diagram

An **association** is a unidirectional relationship that describes a link between objects.

You can create an association between:

- An actor and a use case
- A use case and an actor

Associations in a use case diagram are very close to those in the class diagram. However, it is not possible to specify roles, cardinalities and navigability for associations.

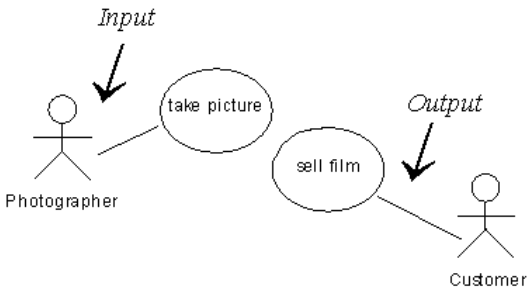
The orientation of the association is defined by the way it is created graphically:

An association between an actor and a use case is an **input** association. An association between a use case and an actor is an **output** association.

The orientation of the association may not be displayed. If you want to display it, you should modify the association display preferences.

For more information on modifying display preferences, see section Modifying association display preferences in a use case diagram.

Example



Association properties in a use case diagram

You can double-click any association symbol in a diagram to display its properties. An association has the following general properties:

Property	Description	Maximum length
Name	Name of the association	254
Code	Reference name or code for the association	254
Comment	Descriptive comment for the association	—
Stereotype	Extends the semantics of an association derived from existing associations but specific to your needs	254
Orientation	Defines the direction of the association, provided the Show Direction option is selected in the display preferences	—
Source	Object the association starts from. You can click the Properties tool beside this box to display the object property sheet	—
Destination	Object the association ends on. You can click the Properties tool beside this box to display the object property sheet	—

Creating an association in a use case diagram

There are several ways to create an association:

- From a diagram
- From the Browser

At creation, an association has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create an association in a use case diagram, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an association from a use case diagram:**

1 Click the Association tool.

2 Click inside an actor and while continuing to hold down the mouse button, drag the cursor to a use case. Release the mouse button inside the use case. You can also create an association from a use case to an actor.

The association link appears between both objects.

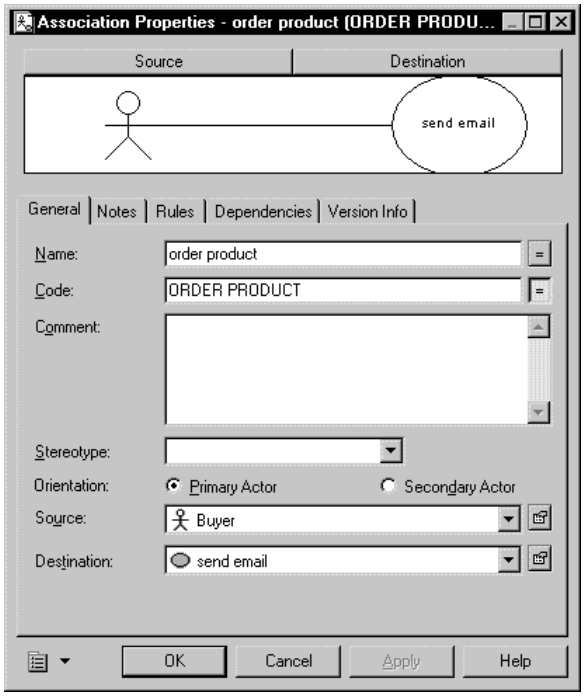
> **Dragging an association to a different object**
> You move the association to a different object by clicking the association line to select it, pressing down CTRL, and dragging one of the attach points to a different object.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Association tool.

4 Double-click the association to display its property sheet.

5 Type a name and a code.



You can open the property sheet of the source and destination objects by clicking the Source and Destination buttons located in the upper part of the association property sheet.

6 Click OK.

Modifying association properties in a use case diagram

You can modify the association properties using the following methods:

- From the association property sheet
- From the List of Associations

For more information on the different ways to modify the association properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying association display preferences in a use case diagram

You can modify the following display preferences for associations using the Tools→Display Preferences command:

Preference	Description
Show name	Displays the name of the association
Show stereotype	Displays the stereotype of the association
Show direction	Displays the orientation of the association

Show direction

If you want to respect the UML methodology, you can set Show direction as not selected by default. If you do want to display the association direction, you can use the association display preferences to modify it.

If no direction is displayed, the position of actors in the diagram implicitly provides the direction of associations. When an actor is positioned to the left of the use case, the association goes from the actor to the use case; inversely, when an actor is positioned to the right of the use case, the association goes from the use case to the actor.

Defining generalizations in a use case diagram

A **generalization** is a relationship between a more general element (the parent) and a more specific element (the child). The more specific element is fully consistent with the more general element and contains additional information.

A generalization can be created between:

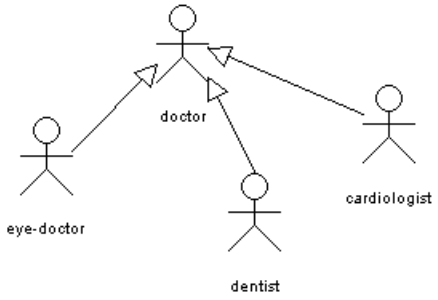
- Two actors
- Two use cases

Generalizations in a use case diagram are similar to those in the class or component diagrams. You create a generalization relationship when several actors, or several use cases have common behaviors.

You can also create a generalization between a shortcut of an actor and an actor, or between a shortcut of use case and a use case. Whenever a link is oriented, only the parent object can be the shortcut.

Example

For example two or more actors may have similarities, and communicate with the same set of use cases in the same way. This similarity is expressed with generalization to another actor. In this case, the child actors inherit the roles, and relationships to use cases held by the parent actor. A child actor includes the attributes and operations of its parent.



Generalization properties in a use case diagram

You can double-click any generalization symbol in a diagram to display its properties.

If the generalization is created in a use case diagram, you cannot change the type of objects linked by the generalization. For example, you cannot attach the dependency coming from a use case to a class, or an interface.

A generalization has the following general properties:

Property	Description	Maximum length
Name	Name of the generalization	254
Code	Reference name or code for the generalization	254
Comment	Descriptive comment for the generalization	—
Parent	Parent actor or use case. You can click the Properties tool beside this box to display the parent property sheet	—
Child	Child actor or use case. You can click the Properties tool beside this box to display the child property sheet	—
Stereotype	Extends the semantics of a generalization derived from existing generalizations but specific to your needs. <<implementation>> means that the child object inherits the implementation of the parent object but does not make its interfaces public, nor support them, thus violating its substitutability	254
Visibility	Visibility of the generalization, whose value denotes how it is seen outside its enclosing namespace	—

Visibility

Property	Visible
Private	Only to the generalization itself
Protected	Only to the generalization and its inherited objects
Package	To all objects contained within the same package
Public	To all objects (option by default)

Creating a generalization in a use case diagram

There are several ways to create a generalization:

- From a diagram
- From the Browser

At creation, a generalization has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a generalization, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a generalization from a use case diagram:**

1 Click the Generalization tool.

2 Click inside an actor and while continuing to hold down the mouse button, drag the cursor to another actor. Release the mouse button inside the second actor. You can also create the generalization between two use cases.

The generalization link appears between both objects.

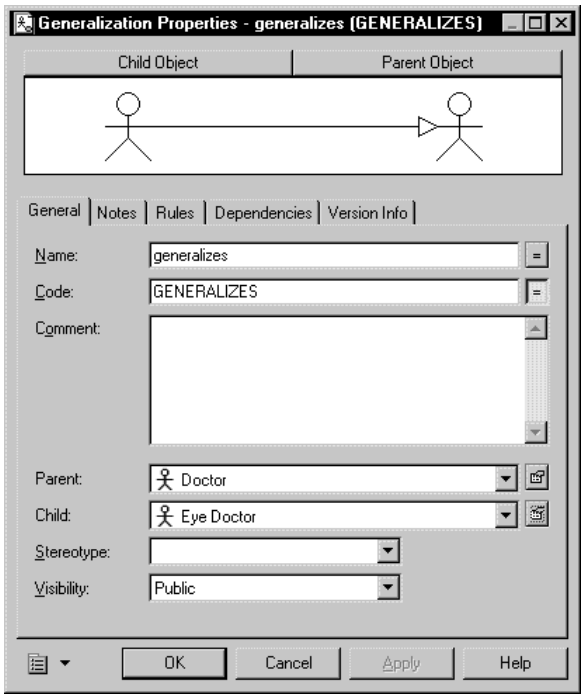
> **Dragging a generalization to a different object**
> You move the generalization to a different object by clicking the generalization line to select it, pressing down CTRL, and dragging one of the attach points to a different object.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Generalization tool.

4 Double-click the generalization to display its property sheet.

5 Type a name and a code.



You can open the property sheet of the child and parent objects by clicking the Child Object and Parent Object buttons located in the upper part of the generalization property sheet.

6 Click OK.

Modifying generalization properties in a use case diagram

You can modify the generalization properties using the following methods:

- From the generalization property sheet
- From the List of Generalizations

For more information on the different ways to modify the generalization properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying generalization display preferences

You can modify the following display preferences for generalizations using the Tools→Display Preferences command.

Preference	When selected
Show name	Displays the name of the generalization
Show stereotypes	Displays the stereotypes of the generalization
Show constraints	Displays the constraints (business rules) of the generalization

Defining dependencies in a use case diagram

A dependency is a semantic relationship between two modeling elements, in which a change to one modeling element (the influent element) may affect the semantics of the other modeling element (the dependent element).

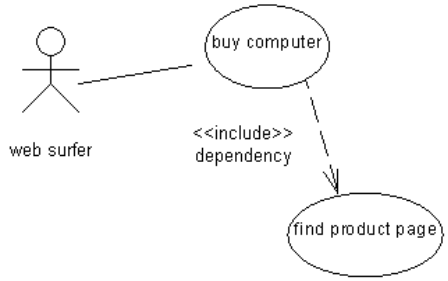
A dependency can be created between:

- An actor and a use case (and vice versa)
- Two actors
- Two use cases

A dependency created between two use cases, or two actors, means that the former object "uses" the latter, in other words, the behavior defined for one object employs the behavior defined for the other.

Example

Buying a computer from a web site involves the activity of finding the product page within the seller's web site:



Dependency properties in a use case diagram

You can double-click any dependency symbol in a diagram to display its properties.

If the dependency is created in a use case diagram, you cannot change the objects linked by the dependency. For example, you cannot attach the dependency coming from a use case to a class or an interface.

A dependency has the following general properties:

Property	Description	Maximum length
Name	Name of the dependency	254
Code	Reference name or code for the dependency	254
Comment	Descriptive comment for the dependency	—
Influent	Selected use case or actor influences the dependent object. Changes on the influent object affect the dependent object. You can click the Properties tool beside this box to display the object property sheet	—
Dependent	Selected use case or actor depends on the influent object. Changes on the dependent object do not affect the influent object. You can click the Properties tool beside this box to display the object property sheet	—
Stereotype	Extends the semantics of a dependency derived from existing dependencies but specific to your needs	—

Stereotype

You can type the following common values for stereotypes of dependencies:

Stereotype	Description
<<extend>>	Relationship specifying how the behavior defined for the first object can be inserted into the behavior defined for the second object
<<include>>	Inclusion of the behavior of the first object into the behavior of the client object, under the control of the client object

Creating a dependency in a use case diagram

There are several ways to create a dependency:

- From a diagram
- From the Browser

At creation, a dependency has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a dependency, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a dependency from a use case diagram:**

1 Click the Dependency tool.

2 Click inside a use case, or an actor and while continuing to hold down the mouse button, drag the cursor to another use case or actor. Release the mouse button inside the second object.

The dependency link appears between both objects.

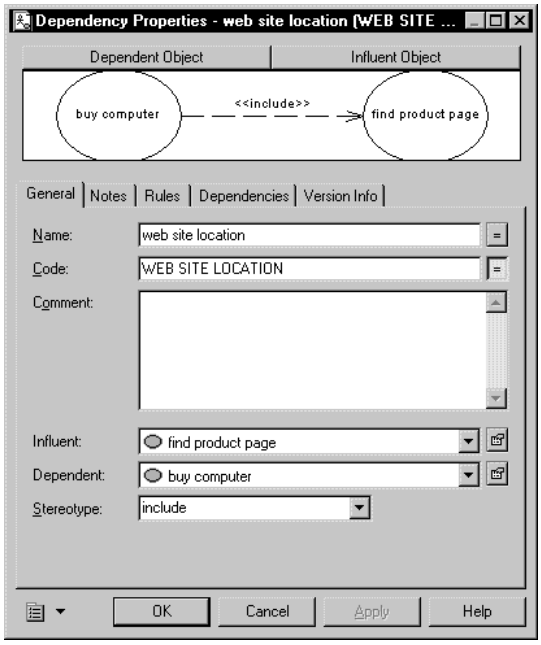
> **Dragging a dependency to a different object**
> You move the dependency to a different object by clicking the dependency line to select it, pressing down CTRL, and dragging one of the attach points to a different object.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Dependency tool.

4 Double-click the dependency to display its property sheet.

5 Type a name and a code.



You can open the property sheet of the dependent and influent objects by clicking the Dependent Object and Influent Object buttons located in the upper part of the dependency property sheet.

6 Click OK.

Modifying dependency properties in a use case diagram

You can modify the dependency properties using the following methods:

- From the dependency property sheet
- From the List of Dependencies

For more information on the different ways to modify the dependency properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying dependency display preferences

You can modify the following display preferences for dependencies using the Tools→Display Preferences command.

Preference	When selected
Show name	Displays the name of the dependency
Show stereotype	Displays the stereotype of the dependency
Show constraints	Displays the constraints (business rules) of the dependency

CHAPTER 3

Building a Class Diagram

About this chapter

This chapter describes how to build a class diagram. It explains the role of each element in a class diagram and how to create and modify them.

Contents

Topic	Page
Class diagram basics	64
Defining classes	65
Defining interfaces	96
Defining attributes in a class diagram	105
Defining identifiers	116
Defining operations	120
Defining associations in a class diagram	139
Defining generalizations in a class diagram	151
Defining dependencies in a class diagram	156
Defining realizations	161
Defining domains in an OOM	165
Defining check parameters	170

Class diagram basics

A **class diagram** defines the static structure of the model, it contains a set of packages, classes, interfaces, and their relationships. These objects together form a class structure that is the logical design view of all, or part of a software system.

Defining a class diagram

The class diagram is one of the diagrams of the Unified Modeling Language (UML). It is a static design view of a system.

The class diagram is part of an Object-Oriented Model. It is one of the available diagrams in an OOM. It shows the symbols of the objects defined in the system. The objects are the classes, packages, and the relationships among them.

Why build a class diagram?

You build a class diagram to simplify the interaction of objects in the system you are modeling. Class diagrams express the static structure of a system in terms of classes and relationships between those classes. A class describes a set of objects, and an association describes a set of links; objects are class instances, and links are association instances.

A class diagram does not express anything specific about the links of a given object, but it describes, in an abstract way, the potential link from an object to other objects.

For more information on diagrams of the OOM, see section What is an OOM, in chapter Object-Oriented Model Basics.

Creating a class diagram

You can create a class diagram using different methods.

For more information on creating a class diagram in a new OOM, see section Creating an OOM, in chapter Object-Oriented Model Basics.

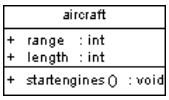
For more information on creating a class diagram in an existing OOM, see section Creating a new diagram, in chapter Managing Models, in the *General Features Guide*.

Defining classes

A **class** is a description of a set of objects that have a similar structure and behavior, and share the same attributes, operations, relationships, and semantics. Structure is described by attributes and associations. Behavior is described by operations. A class usually implements one or more interfaces.

Classes are the main building blocks of an OOM. Classes, and the relationships that you create between them, form the basic structure of an OOM. A class defines a concept within the application being modeled, such as a physical thing (like a car), a business thing (like an order) a logical thing (like a broadcasting schedule), an application thing (like an OK button), or a behavioral thing (like a task).

The following example shows the class Aircraft with its attributes (range and length) and operation (startengines).



From the property sheet of a class you access the following information about the class:

- Attributes. These are named properties of the class that describe its characteristics. See section Defining attributes in a class diagram.
- Identifiers. These are class attributes, or combinations of class attributes, whose values uniquely identify each occurrence of the class. See section Defining identifiers.
- Operations. These are specification of a query that an interface may be called to execute. See section Defining operations.
- Associations. These display the list of migrated attributes proceeding from navigable associations connected to the class. See section Viewing the migrated attributes of a class.
- Inner classifiers. These display the list of inner classes and interfaces of the current class. See section Defining inner classifiers.
- Script. This allows you to customize your scripts. See section Customizing scripts for classes and interfaces.
- Preview. This allows you to preview the code of the class. See section Previewing the code of a class or interface.

Class properties

You can double-click any class symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the class	254
Code	Reference name of the class	254
Comment	Descriptive comment for the class	—
Stereotype	Sub-classification derived from an existing class. It lets you extend the semantics of an object without changing its structure and can be predefined or user-defined	254
Type	Category used for implementation purpose	—
Visibility	Visibility of the class, its value denotes how it is seen and used by other objects	—
Cardinality	Specific numbers of instances that the class can have, for example: 0...1	—
Abstract	Indicates that the class cannot be instantiated and therefore has no direct instances	—
Final	The class cannot have any inherited objects	—
Generate	The class is automatically included among the objects generated from the model when you launch the generation process	—
Inner to	Indicates the name of the class or interface to which the current class belongs as an inner classifier	254
Association class	Name of the association related to the class to form an association class. The attributes and operations of the current class are used to complement the definition of the association	254

For more information on other properties specific to Web services, see section Detail page of the class, in chapter Working with Web Services.

A class also includes the following properties:

Property	Description
Attribute	Defines the characteristics of a class. For example, Name and Surname can be two attributes of a class named Employee
Identifier	Class attributes, whose values identify each occurrence of the class
Operation	Carries out a service that effects behavior
Rule	A business rule that your business follows

Persistent class

The Detail page contains a Persistent groupbox whose purpose is to improve the generation of codes and data types during generation of a conceptual data model or a physical data model from an object-oriented model.

The Persistent groupbox contains the following values:

- Code: code of the table or entity that will be generated in a persistent CDM or PDM model. You have to type the code that you want to give to the table or entity in the Code box.

  Example: to generate a class Purchaser into a table PURCH, type **PURCH** in the Code box.

- Abstract Data Type: a user-defined data type that can encapsulate a range of data values and functions. When selected, indicates that the class should be generated in a PDM as an abstract data type, provided the current DBMS supports Abstract Data Type.

  For more information on abstract data types, see section Using abstract data types, in chapter Building physical diagrams in the *Physical Data Model User's Guide*

For more information on object persistence, see section Managing object persistence during OOM to PDM generation, in chapter Generating from an Object-Oriented Model.

Class stereotype

You can modify an existing stereotype or create a new one from the object language. You can also use the extended model definition to create new sets of stereotypes.

For more information on how to manipulate categories and entries in the object language, see chapter The Resource Editor in the *General Features Guide*.

For more information on extended model definitions, see section Working with extended model definitions, in chapter Managing Objects in the *General Features Guide*.

Default stereotypes

You can declare a class to be one of the following default stereotypes:

Stereotype	Description
<<actor>>	Coherent set of roles that users play
<<enumeration>>	List of named values used as the range of an attribute type
<<exception>>	Exception class, mainly used in relation to error messages
<<implementationClass>>	Class whose instances are statically typed. Defines the physical data structure and methods of a class as implemented in traditional programming languages
<<process>>	Heavyweight flow that executes concurrently with other processes
<<signal>>	Specification of asynchronous stimulus between instances
<<metaclass>>	Stereotyped classifier denoting that the class is a metaclass of some other class
<<powertype>>	Stereotyped classifier denoting that the classifier is a metaclass whose instances are subclasses of another class
<<thread>>	Lightweight flow that executes concurrently with other threads within the same process. Usually executes inside the address space of an enclosing process
<<type>>	Abstract class used to specify the structure and behavior of a set of objects but not the implementation
<<utility>>	Class that has no instances

Other language stereotypes

Depending on the object language attached to your model, more stereotypes appear in the list of stereotypes. These are used to specify the way classes should be generated in the selected object language.

These stereotypes are defined in the object language file.

For more information on the stereotypes of other languages, see section Stereotypes category, in chapter Object Languages Reference Guide in the *Advanced User Documentation.*

Class type

A type is used to define the class implementation. For example, you can use the type to implement a task (middle tier, business object), or a storage type.

You can declare a class to be one of the following types:

- Business Object
- Class
- Storage
- Utility
- Visual Object
- JavaBean

Class visibility

The visibility of a class is the way it can be seen by other objects. When a class is visible to another object, it may influence the structure or behavior of the object, or similarly, the other object can affect the properties of the class.

Property	Visible
Private	Only to the class itself
Protected	Only to the class and its derived objects
Package	To all objects contained within the same package
Public	To all objects in the model

Class cardinality

The cardinality of a class specifies the number of instances a class can have.

Cardinality	Number of instances
0..1	None to one
0..*	None to an unlimited number
1..1	One to one
1..*	One to an unlimited number
*	Unlimited number

Class related diagrams

In the property sheet of a class, you can select the Related Diagrams tab to display all diagrams of the model that are related to a class. They can be attached and detached from the current class at this stage. This page displays a list of the diagrams identified by their name and code.

For more information on related diagrams, see section Defining related diagrams, in chapter Working with Object-Oriented Models.

Creating a class

There are several ways to create a class:

- From a diagram
- From the Browser
- From the list of classes

At creation, a class has a default name including a number, this number is assigned in the order of creation.

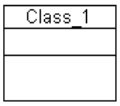
For more information on the different ways to create a class, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a class from a diagram:**

1 Click the Class tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:

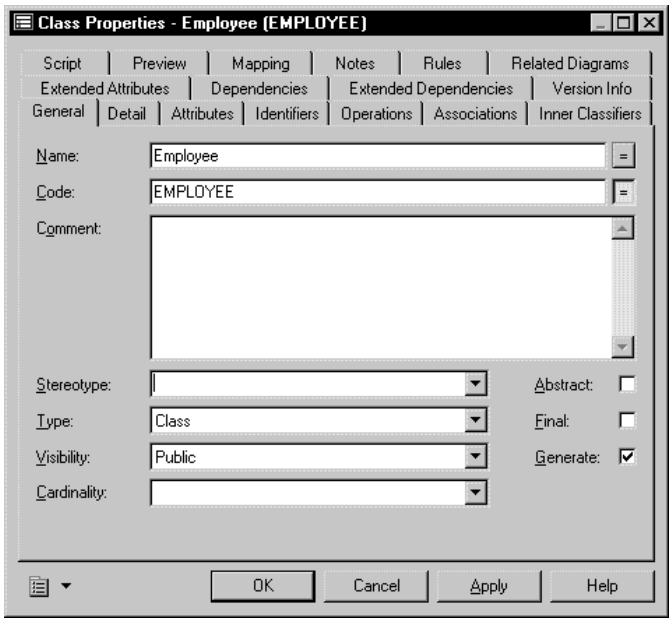


3 Click the Pointer tool.
or
Click the right mouse button.

You release the Class tool.

4 Double-click the class symbol to display the class property sheet.

5 Type a name and a code.



6 Click OK.

Creating a class from an interface

You can create a class in a diagram from the contextual menu of an interface. This allows you to inherit all the operations of the interface, including the getter and setter operations.

When you create a class this way, a realization link is automatically created between the class and the interface. Moreover, the class property sheet displays the realization link in the Realizes tabbed page of the Dependencies page.

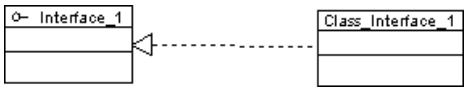
❖ **To create a class from an interface:**

1 Right-click an interface in the diagram to display the interface contextual menu.

2 Select Create Class from the contextual menu.

A new class and a new realization are created in the diagram. They are also visible from the Browser.

The class is suffixed by the name of the interface.



Defining Java BeanInfo classes

A JavaBean is a reusable software component written in Java that can be manipulated visually in a builder tool. A Java BeanInfo class is used as a standard view of a Bean. Each JavaBean can implement a BeanInfo class. Bean implementors may want to provide explicit information about the methods, properties, and events of a Bean by providing a Java BeanInfo class.

The BeanInfo class is generated with an attribute, and the following operations:

- constructor
- getPropertyDescriptors();
- getMethodDescriptors();

You can view the complete code by clicking the Preview tab in the BeanInfo class property sheet.

Attribute created

The attribute has the following code:

```
private static final Class <ClassCode>Class =
<ClassCode>.class;
```

Operations created

The constructor has the following code:

```
<ClassCode>BeanInfo()
{
     super();
   }
```

The getPropertyDescriptors() operation has the following code:

```
public PropertyDescriptor[] getPropertyDescriptors ()
{
 // Declare the property array
 PropertyDescriptor properties[] = null;

 // Set properties
 try
 {
    // Create the array
    properties = new PropertyDescriptor[<nbProperties>];
    // Set property 1
    properties[0] = new
PropertyDescriptor("<propertyCode1>" ,<ClassCode>Class;
    properties[0].setConstrained(false);
    properties[0].setDisplayName("propertyName1");

properties[0].setShortDescription("propertyComment1");
    // Set property 2
    properties[1] = new
PropertyDescriptor("<propertyCode2>" ,<ClassCode>Class;
    properties[1].setConstrained(false);
    properties[1].setDisplayName("propertyName2");

properties[1].setShortDescription("propertyComment2");

 }
 catch
 {
    // Handle errors
 }
 return properties;
}
```

The getMethodDescriptors() operation has the following code:

```
public MethodDescriptor[] getMethodDescriptors ()
{
 // Declare the method array
 MethodDescriptor methods[] = null;
 ParameterDescriptor parameters[] = null;

 // Set methods
 try
 {
    // Create the array
    methods = new MethodDescriptor[<nbMethods>];
    // Set method 1
    parameters = new
ParameterDescriptor[<nbParameters1>];
    parameters[0] = new ParameterDescriptor();
    parameters[0].setName("parameterCode1");
    parameters[0].setDisplayName("parameterName1");

parameters[0].setShortDescription("parameterComment1");
    methods[0] = new MethodDescriptor("<methodCode1>",
parameters);
    methods[0].setDisplayName("methodName1");
    methods[0].setShortDescription("methodComment1");
    // Set method 2
    methods[1] = new MethodDescriptor("<methodCode2>");
    methods[1].setDisplayName("methodName2");
    methods[1].setShortDescription("methodComment2");

 }
 catch
 {
    // Handle errors
 }
 return methods;
 }
```

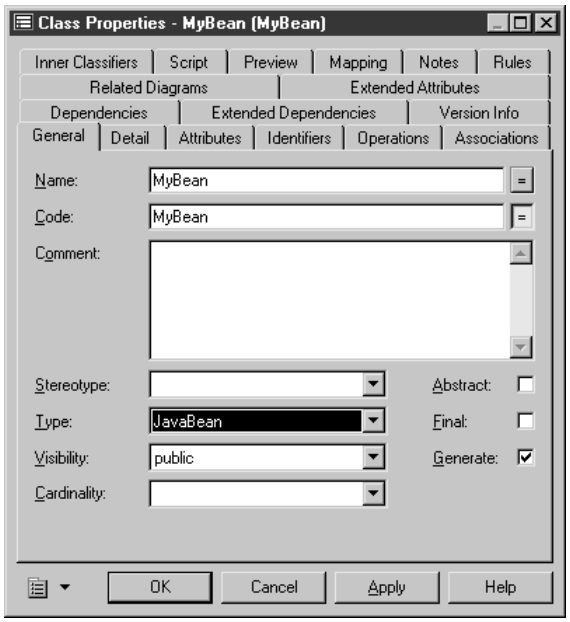
Creating Java BeanInfo classes

If you are using the Java object language, you can create Java BeanInfo classes from the classes in an OOM using the following methods:

- Creating a Java BeanInfo class from the Language menu
- Creating a Java BeanInfo class from the class contextual menu

A BeanInfo class can only be created from a class if its type is JavaBean. This type is only available if the current object language is Java.

You can define the type of a class from its property sheet:



When you create a Java BeanInfo class, a dependency link is automatically created between both classes. Moreover, the Java BeanInfo class property sheet displays the dependency link in the Depends on tabbed page of the Dependencies page.

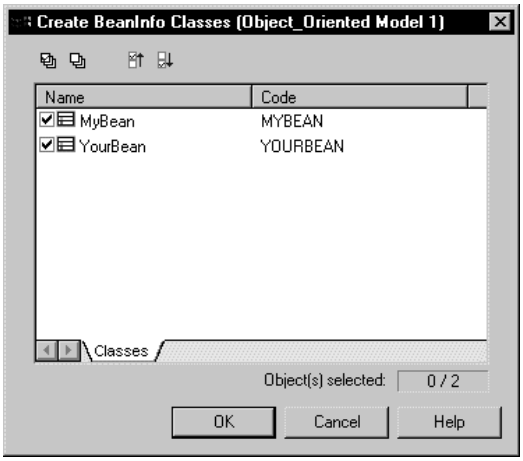
After creation, the stereotype of the Java BeanInfo class automatically switches to the <<BeanInfo>> stereotype.

Creating a Java BeanInfo class from the Language menu

PowerDesigner generates a new BeanInfo class for each selected class. You can select any of the classes from the model, including those that are contained within packages.

❖ **To create a Java BeanInfo class from the Language menu:**

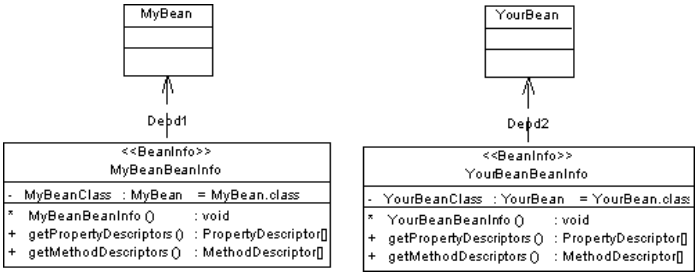
1 Select Language→Create BeanInfo Classes to display the Create BeanInfo Classes selection window. It contains a list of all the classes of type JavaBean in the model.



2 Select the classes for which you want to generate Java BeanInfo classes.

3 Click OK.

A BeanInfo class is created in the model for each selected class.



Creating a Java BeanInfo class from the class contextual menu

❖ **To create a Java BeanInfo class from the class contextual menu:**

1 Right-click a class in the diagram to display the class contextual menu.

2 Select Create BeanInfo Class from the contextual menu.

The new class and a new dependency are created in the diagram. They are also visible from the Browser. The Java BeanInfo class is prefixed after the influent class.

Classifiers

A classifier, in UML terminology, is a model element with structural (attributes) and behavioral (operations) features. A class is the most important classifier, but interfaces, or use cases for example are also classifiers.

Defining inner classifiers

It is possible to declare a class within another class or interface by creating an **inner classifier**. Inner classifiers help you improve the overall visibility of your model by allowing you to group together classes or interfaces that logically belong together. A class declared within another class or interface belongs to the namespace of this class or interface.

You can click the Inner Classifiers tab in the class or interface property sheet to display inner classifiers of the current class or interface.

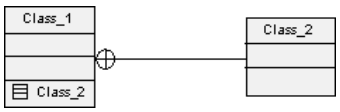
You can use inner classifiers during generation and reverse engineering into the following languages:

- Java
- XML
- PowerBuilder (you can declare a class within another class)

Defining inner links

When you declare a class within another class or interface, you create an **inner link** between both classifiers. It is created between a parent class and a child class indicating that the child class is defined and visible only in the parent class. Inner links are used for inner classifiers' relationships only.

The symbol of the inner link is a line with an anchor icon near the parent class symbol. The anchor icon is a cross inside a circle.



There is no property sheet of inner link, nor is it accessible from the Browser. When you double-click an inner link symbol in the diagram, the property sheet of the parent classifier opens. If you want to change the inner classifier, you have to select the Inner Classifiers tab to select the inner classifier of your choice.

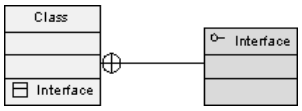
Using models from previous versions
If you use a model that has been created in a previous version, an inner link without symbol is created for each inner classifier. The <<inner>> dependency between the inner classifier and the parent classifier is removed.

❖ **To create an inner link:**

1 Select the Inner Link tool from the palette.

2 Click inside the parent classifier and while continuing to hold down the mouse button, drag the cursor to the child classifier. Release the mouse button inside the child classifier.

The inner link appears between both classifiers. If you have selected the Show Inner Classifiers display preference in the class or interface category, the name of the child classifier appears in the symbol of the parent classifier.



3 Click the pointer tool.
or
Click the right mouse button.

You release the Inner Link tool.

4 Double-click the inner link in the diagram to display the property sheet of the parent classifier.

5 Modify classifier properties if required.

6 Click OK.

Attaching an inner classifier to a class or an interface

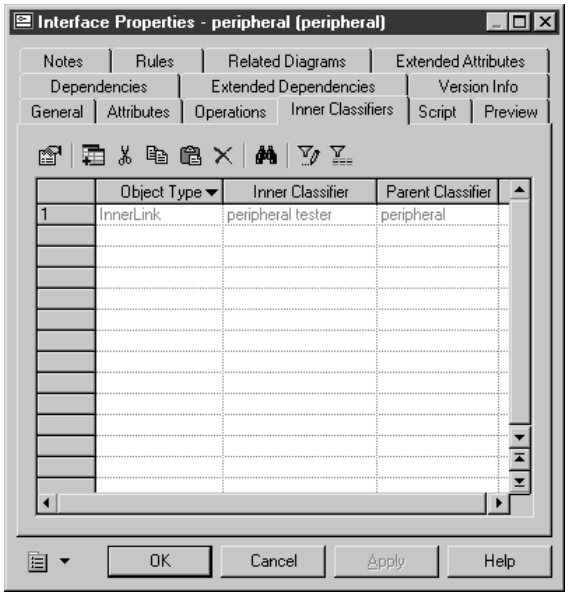
If you do not use the Inner Link tool in the palette, you can also attach an inner classifier to a class or interface from the Inner Classifiers page of the class or interface property sheet.

Attaching an inner classifier to another automatically creates an inner link between them.

❖ To attach an inner classifier to a class or an interface:

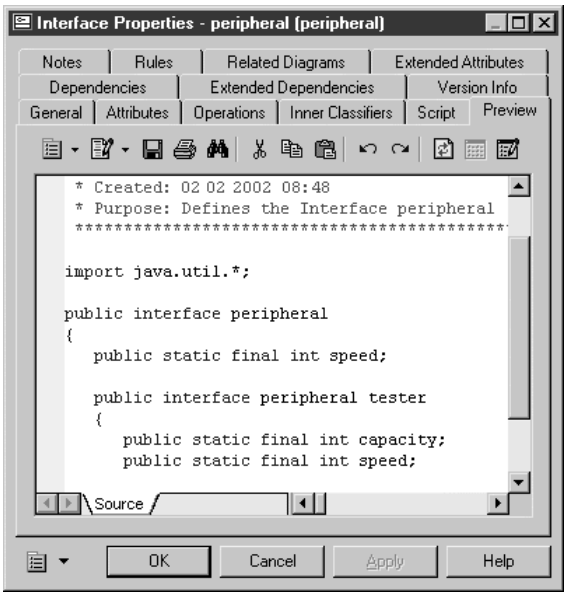
1. Double-click a class or an interface in the diagram to display the property sheet.
2. Click the Inner Classifiers tab to display the Inner Classifiers page.
3. Click the Attach Inner Classifier tool.

   A selection window appears.
4. Select the classes or interfaces you want to attach as inner classifiers to the current class or interface.
5. Click OK.



The classes or interfaces appear in the list of inner classifiers, and the inner classifier definitions are added to the current class or interface.

6 Click the Preview tab to visualize the inner classifier definition within the current class or interface.



7 Click OK.

The classifiers are connected by an inner link in the diagram.

Detaching an inner classifier from a class or an interface

You can detach an inner classifier from a class or an interface. This will remove its declaration from the class or the interface.

❖ **To detach an inner classifier from a class or an interface:**

1 Double-click the parent class or interface in the diagram, click the Inner Classifiers tab, and select an inner classifier from the list of inner classifiers.
or
Select the inner link in the diagram.

2 Click the Delete tool.

3 Click OK.

The inner classifier definition is detached from the current class or interface.

Adding objects to a class

The following objects, belonging to a class in the model, can be added to another class:

Object	Description
Attribute	Defines the characteristics of a class
Operation	Implementation of a service that can be requested from any object of the class in order to affect behavior

When you add an object to a class, you create a copy of the object. The new object exists as a unique object, you can then make changes to it as you would to any object in the model.

Adding an attribute to a class

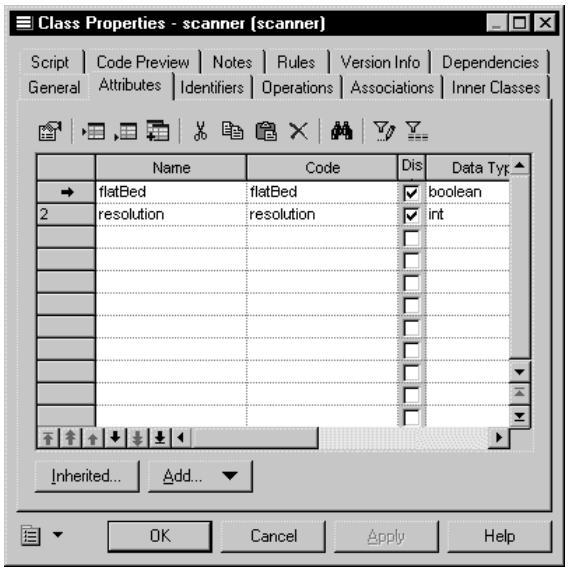
An attribute is a named property of a class defining the characteristics of the class.

You can use the attributes of other classes and add them to the current class.

❖ **To add an attribute to a class:**

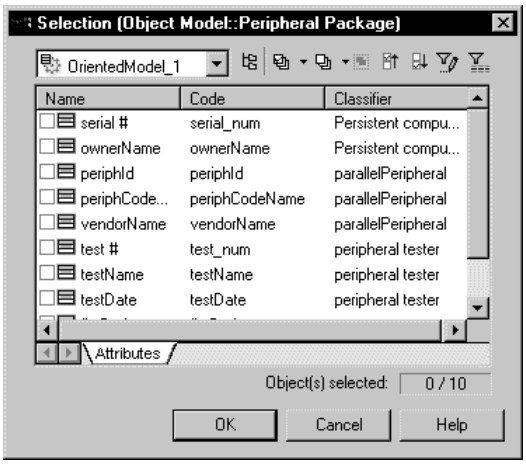
1 Double-click a class in the diagram to display the class property sheet.

2 Click the Attributes tab to display the Attributes page.



3 Click the Add Attributes tool.

The Selection window appears. It contains a list of all the attributes in the model, except those that already belong to the class.



4 Select the attributes you want to add to the class.
or
Use the Select All tool to add all the attributes in the list to the class.

5 Click OK.

The attributes are added to the list of attributes of the current class and appear at the end of the list.

6 Click OK.

Adding an operation to a class

An operation is the implementation of a service that can be requested from any object of the class in order to affect behavior.

You can use the operations of other classes and add them to the current class.

❖ **To add an operation to a class:**

1 Double-click a class in the diagram to display the class property sheet.

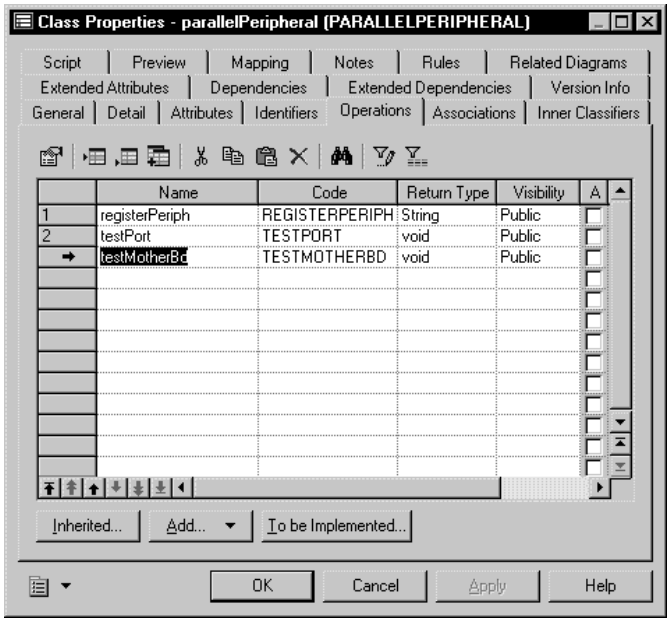
2 Click the Operations tab to display the Operations page.

3 Click the Add Operations tool.

The Selection window appears. It contains a list of all the operations that exist in the model, except those that already belong to the class.

4 Select the operations you want to add to the class.

5 Click OK.



The operations are added to the class and appear at the end of the list.

6 Click OK.

Attaching a classifier to a data type or a return type

A data type or a return type can be a classifier. You can choose to attach a class, an interface, or a classifier from a JDK library for example, to the following objects:

- Attribute data type
- Operation return type
- Parameter data type

The classifier can belong to the current model or to another model. If it belongs to the current model or package, it is displayed by default together with the other classifiers of the current package. If it belongs to another model or package, a shortcut of the classifier is automatically created in the current package.

Fully qualified name

If the classifier belongs to another package of the same model, you can manually attach the classifier to a data type or return type by typing the fully qualified name of the classifier directly in the data type or return type box.

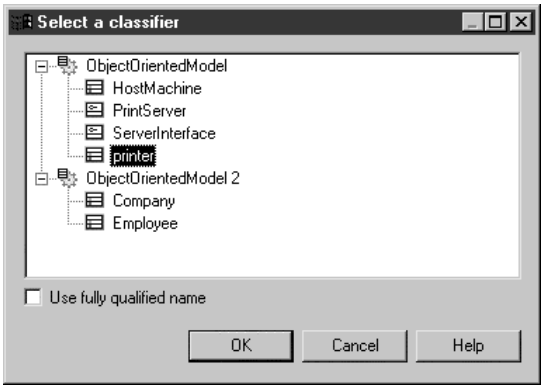
For example, if you need to reference Class_2 as a classifier data type of an attribute named Attr_1 in Class_1, open the Attr_1 property sheet and type **`PACKAGE_2.CLASS_2`** in the Data type dropdown listbox.

Package_1
|__ Class_1
|__ Attr_1

Package_2
|__ Class_2

After you type the fully qualified name (path and classifier name), it is displayed in the data type or return type box.

When a data type or a return type is created, there is no classifier attached to it, that is why the Use fully qualified name check box is not selected in the Select a Classifier window.



However if you type a relative name, like 'Lang.Integer' instead of 'Java.Lang.Integer', the Use fully qualified name check box is automatically selected because a fully qualified name corresponds to this name.

Attaching a classifier to an attribute data type

You can attach a classifier from the attribute property sheet or from the Data Type column in the list of attributes.

❖ **To attach a classifier to an attribute data type:**

1 Double-click a class in the diagram to display the class property sheet.

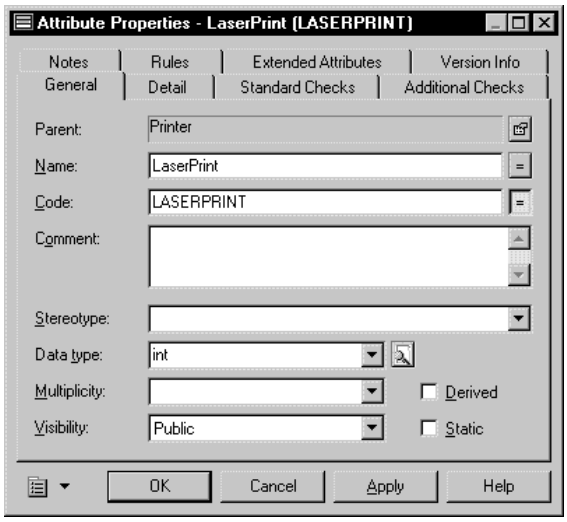
2 Click the Attributes tab to display the Attributes page.

3 Select an attribute in the list.

An arrow appears at the beginning of the line.

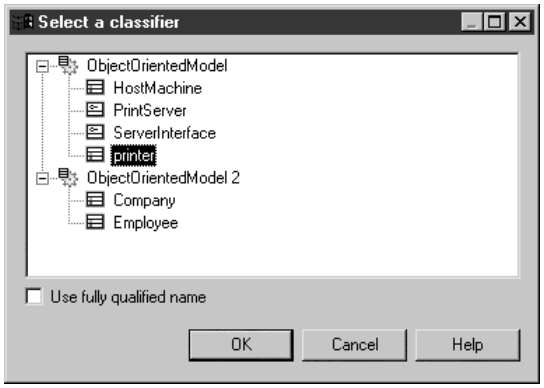
4 Click the Properties tool.
or
Double-click the arrow at the beginning of the line.

The attribute property sheet appears.



5 Click the Select a Classifier button beside the Data type dropdown listbox.

A Select a Classifier window appears. It displays all classifiers of all open models.



6 Double-click the classifier you want to attach to the data type.

You return to the attribute property sheet. The classifier name is now displayed in the Data type dropdown listbox.

7 Click OK.

> **Synchronizing data type and classifier**
> When you select a data type that matches a classifier's code, the data type is automatically synchronized if the classifier's code changes.

Attaching a classifier to an operation return type

You can attach a classifier from the operation property sheet or from the Return Type column in the list of operations.

❖ **To attach a classifier to an operation return type:**

1 Double-click a class in the diagram to display the class property sheet.

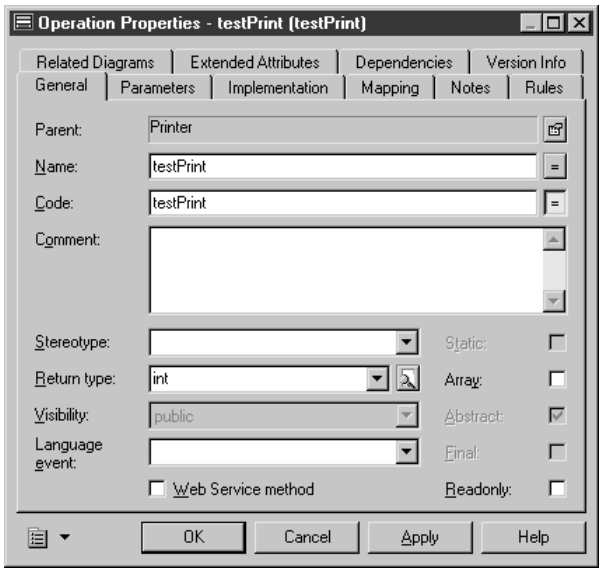
2 Click the Operations tab to display the Operations page.

3 Select an operation in the list.

An arrow appears at the beginning of the line.

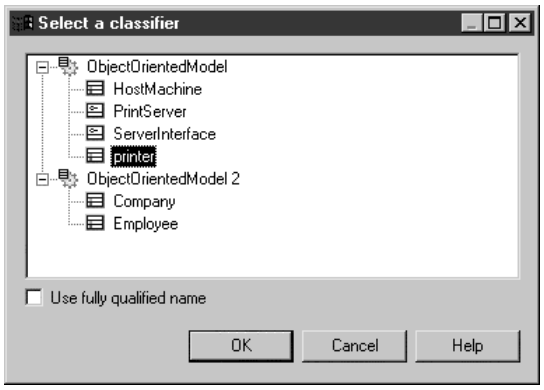
4 Click the Properties tool.
or
Double-click the arrow at the beginning of the line.

The operation property sheet appears.



5 Click the Select a Classifier button beside the Return type dropdown listbox.

A Select a Classifier window appears. It displays all classifiers of all open models.



6 Double-click the classifier you want to attach to the return type.

You return to the operation property sheet. The classifier name is now displayed in the Return type dropdown listbox.

7 Click OK.

Synchronizing return type and classifier
When you select a return type that matches a classifier's code, the return type is automatically synchronized if the classifier's code changes.

Attaching a classifier to a parameter data type

❖ **To attach a classifier to a parameter data type:**

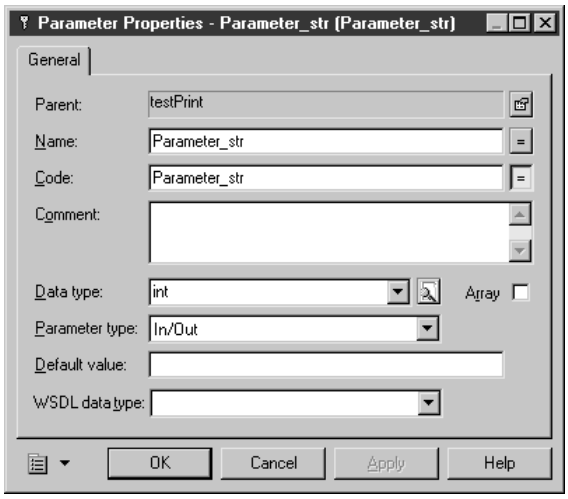
1 Double-click a class in the diagram to display the class property sheet.

2 Click the Operations tab to display the Operations page.

3 Double-click the arrow at the beginning of the line to display the operation property sheet.

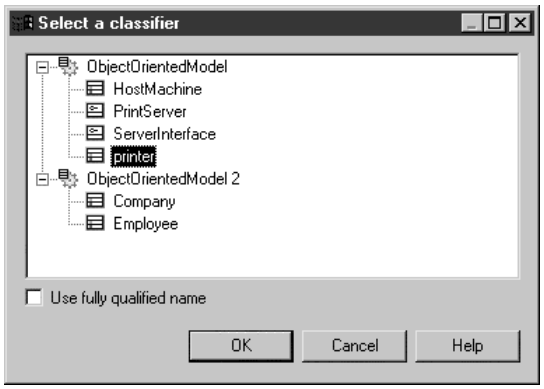
4 Click the Parameters tab to display the Parameters page.

5 Select the parameter you want to modify and click the Properties tool to display the parameter property sheet.



6 Click the Select a Classifier button beside the Data type dropdown listbox.

A Select a Classifier window appears. It displays all classifiers of all open models.



7 Double-click the classifier you want to attach to the data type.

You return to the parameter property sheet. The classifier name is now displayed in the Data type dropdown listbox.

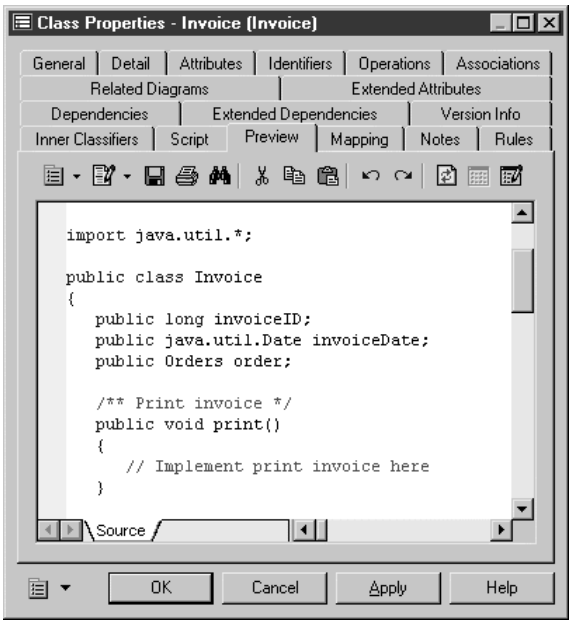
8 Click OK.

Synchronizing data type and classifier
When you select a data type that matches a classifier's code, the data type is automatically synchronized if the classifier's code changes.

Previewing the code of a class or interface

You can preview the code of a class or an interface in order to see their generated code by clicking the Preview tab.

The displayed code changes according to the object language you have selected in your model.



You can use the following tools and accelerators from the Preview toolbar:

Tool	Description	Accelerator
	Open Editor Contextual menu	SHIFT + F11
	Refresh	F5
	Show Generation Options	CTRL + W
	Select Generation Targets	CTRL + F6

Showing generation options

If you click the Show Generation Options tool, the Generation Options dialog box appears. You can change generation options from this dialog box and see the impact on the code.

This feature is especially useful when you are working with Java. For other object languages, generation options do not influence the code.

Select Generation Targets

This tool is available when at least one extended model definition flagged for generation is linked to the model, and when it contains Generated Files entries for the current object. When available, it displays the list of targets for the current object. If you add a generation target, the corresponding tab is added to the Preview page. If you deselect a generation target, the corresponding tab disappears from the Preview page.

For more information on generation targets, see section Generation targets, in chapter Extended Model Definitions Reference Guide, in the *Advanced User Documentation*.

Script bookmarks

In the Preview page, you can add and remove bookmarks at specific points in the code and then navigate forwards or backwards from bookmark to bookmark:

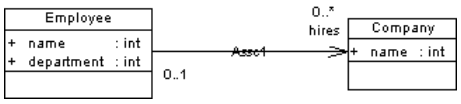
Accelerator	Description
CTRL + F2	Adds a new bookmark. A blue bookmark box appears. If you repeat this action from the same position, the bookmark is deleted and the blue marker disappears
F2	Jumps to bookmark
SHIFT + F2	Jumps to previous bookmark

Note that bookmarks are not printable and are lost if you use the Refresh, or Show Generation Options tools.

Viewing the migrated attributes of a class

Navigable associations migrate attributes to classes during code generation. You can display these migrated attributes in the Associations page of a class property sheet.

In the following example, the class Employee is associated with the class Company.



If you preview the generated code of the class Employee, you can see the following three attributes (in Java language):

```
public class EMPLOYEE
{
   public COMPANY hires[];
   public int NAME;
   public int DEPARTMENT;
}
```

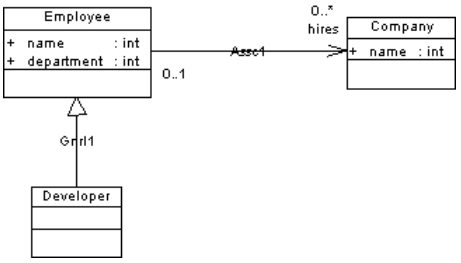
The association between Employee and Company is migrated as the attribute **`public COMPANY hires []`**.

You can use the Associations page of a class property sheet to display the list of all migrated attributes proceeding from navigable associations.

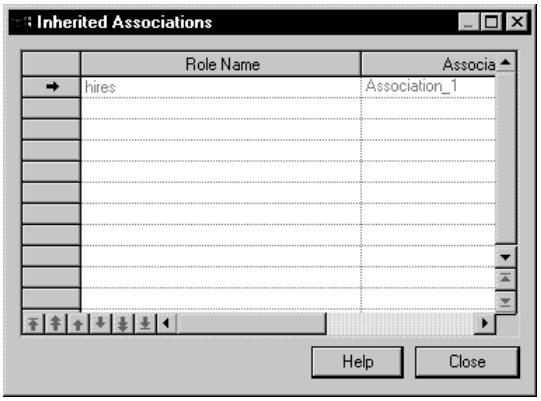
Inherited Associations

When a class inherits from a parent class through a generalization link, you can use the list of Inherited Associations to view the list of migrated attributes from the parent of the current class.

For example, the class Developer inherits from Employee through a generalization.



If you double-click the class Developer, click the Associations tab in the class property sheet and click the Inherited button, the list of Inherited Associations displays the migrated attribute of the class Employee.



For more information on associations, see section Defining associations in a class diagram.

Customizing scripts for classes and interfaces

You customize scripts from the Script page of a class or an interface property sheet. Customizing a creation script allows you to add descriptive information about a generated script.

Examples

If a development project archives all the creation scripts that are generated, a header script can be inserted before each creation script, indicating the date, time, and any other information specific to the generated script.

If an organization requires that generated scripts are filed using a naming system which may be independent from a script name, a header script could direct a generated script to be filed under a different name than the name indicated in the creation script.

Scripts can be customized as follows:

- Insert scripts at the beginning (Header page) and the end (Footer page) of a script
- Insert scripts before and after a class or interface creation command (Imports page)

Script toolbar properties

You can use the following tools and accelerator from the script toolbar:

Tool	Description	Accelerator
	Open Editor Contextual menu	SHIFT + F11
	Edit With. Opens the default editor you previously defined or allows you to select another editor if you click the down arrow beside this tool	CTRL + E
	Import Folder. Available in the Imports tabbed page only. Opens a selection window to select packages to import. The packages are added where the cursor is positioned. They are prefixed by the keyword 'import'	—

Tool	Description	Accelerator
	Import Classifier. Available in the Imports tabbed page only. Opens a selection window to select classifiers to import. The classifiers are added where the cursor is positioned. They are prefixed by the keyword 'import'	—

For more information on defining a default editor, see section Defining a text editor, in chapter Using the PowerDesigner Interface in the *General Features Guide*.

Formatting variables in customized scripts

Variables have a syntax that can force a format on their values in the following ways:

- Force values to lowercase or uppercase characters
- Truncate the length of values

When a variable does not specify formatting, its values have the same format as in the OOM. You can use the following formatting syntax with variables:

Format code	Format of variable value in script
.L	Lowercase characters
.T	Removes blank spaces
.U	Uppercase characters
.n	Maximum length where n is the number of characters
*.n*J	Justifies to fixed length where n is the number of characters

You embed formatting options in variable syntax as follows:

%.*format*:*variable***%**

Modifying class properties

You can modify the class properties using the following methods:

- From the class property sheet
- From the list of classes

For more information on the different ways to modify the class properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying class display preferences

You can modify the following display preferences for classes using the Tools→Display Preferences command:

Preference	Description
Show attributes	Displays all the class attributes, or limits the number displayed to a maximum that you specify in the Limit box
Show operations	Displays all the class operations, or limits the number displayed to a maximum that you specify in the Limit box
Show stereotype	Displays the stereotype of the class
Show constraints	Displays the constraints (business rule) attached to the class
Show inner classifiers	Displays inner classifiers attached to the classes in the model

Defining interfaces

An interface is similar to a class but it is used to implement the specification of an abstraction of a class. It is a collection of operations specifying the externally visible behavior of a class. It has no implementation of its own.

An interface includes the signatures of the operations. It specifies only a limited part of the behavior of a class. A class can implement one or more interfaces.

A class that implements all the operations in an interface is said to realize the interface. A class that requires one or more operations in an interface is said to use the interface.

The following example shows an interface (Name) realizing the action (GetName) for a class (EmployeeInfo).


EmployeeInfo
+ status : boolean
+ ss_number : String
+ salary : int
+ Start_date : Date
+ EmployeeInfo () : String inPhone, String i
GetName
Name
+ firstName : String
+ lastName : String
+ Name () : String inFirstName, String in L

From the property sheet of an interface you access the following information about the interface:

- Attributes. These are named properties of the interface. See section Defining attributes in a class diagram.
- Operations. These are specification of a query that an interface may be called to execute. See section Defining operations.
- Inner classifiers. These display the list of inner classes and interfaces of the current interface. See section Defining inner classifiers.
- Script. This allows you to customize your scripts. See section Customizing scripts for classes and interfaces.
- Preview. This allows you to preview the code of the interface. See section Previewing the code of a class or interface.

Interface properties

You can double-click any interface symbol in a diagram to display its properties.

An interface has the following properties:

Property	Description	Maximum length
Name	Name of the interface	254
Code	Reference name of the interface	254
Comment	Descriptive comment for the interface	—
Stereotype	Sub-classification derived from an existing interface. It lets you extend the semantics of an object without changing its structure and can be predefined or user-defined	254
Visibility	Visibility of the interface, its value denotes how it is seen outside its enclosing name space	—
Inner to	Indicates the name of the class or interface to which the current interface belongs as an inner classifier	254
Generate	The interface is automatically included among the objects generated from the model when you launch the generation process	—

An interface also includes the following properties:

Property	Description
Attribute	Defines a constant property of an interface. An attribute of interface is slightly different from an attribute of class because an interface can only have constant attributes (static and frozen). For example, consider an attribute named RED that is used to define the RGB value of the red color. When this attribute is added to the interface, it is always static and frozen (in C#, for example)
Operation	Carries out a service that effects behavior
Rule	A business rule that your business follows

Interface stereotype

You can declare an interface to be one of the following stereotypes:

Stereotype	Description
<<metaclass>>	Defines the interface that will interact with a model that contains classes with metaclass stereotypes
<<powertype>>	Stereotyped classifier denoting that the classifier is a metaclass whose instances are subclasses of another class
<<process>>	Heavyweight flow that can execute concurrently with other processes
<<thread>>	Lightweight flow that can execute concurrently with other threads within the same process. Usually executes inside the address space of an enclosing process
<<utility>>	Refers to a class that has no instances

Interface visibility

The visibility of an interface is the way it can be seen by other objects. If an interface is visible to another object, it may influence the structure or behavior of the object, or similarly, the other object can affect the properties of the interface.

Property	Visible
Private	Only to the interface itself
Protected	Only to the interface and its derived objects
Package	To all objects contained within the same package
Public	To all objects in workspace

Interface related diagrams

In the property sheet of an interface, you can select the Related Diagrams tab to display all diagrams of the model that are related to an interface. They can be attached and detached from the current interface at this stage. This page displays a list of the diagrams identified by their name and code.

For more information on related diagrams, see section Defining related diagrams, in chapter Working with Object-Oriented Models.

Creating an interface

There are several ways to create an interface:

- From a diagram
- From the Browser
- From the list of interfaces

At creation, an interface has a default name including a number, this number is assigned in the order of creation.

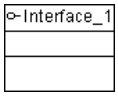
For more information on the different ways to create an interface, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an interface from a diagram:**

1 Click the Interface tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:

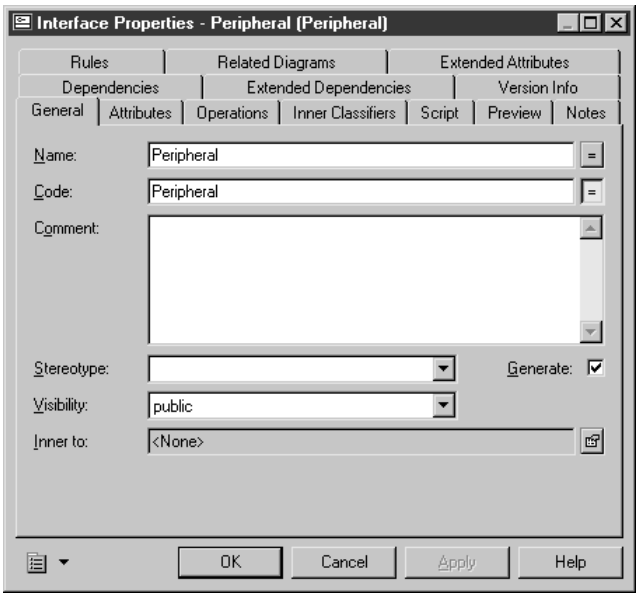


3 Click the Pointer tool.
or
Click the right mouse button.

You release the Interface tool.

4 Double-click the interface symbol to display the interface property sheet.

5 Type a name and a code.



6 Click OK.

Creating an interface from a class

You can create an interface in a diagram from the contextual menu of a class. This allows you to inherit all the operations of the class, including the getter and setter operations.

When you create an interface this way, a realization link is automatically created between the class and the interface. Moreover, the interface property sheet displays the realization link in the Realized by tabbed page of the Dependencies page.

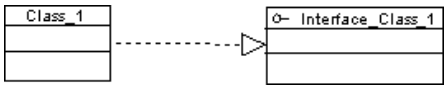
❖ **To create an interface from a class:**

1 Right-click a class in the diagram to display the class contextual menu.

2 Select Create Interface from the contextual menu.

A new interface and a new realization are created in the diagram. They are also visible from the Browser.

The interface is suffixed after the name of the class.



Adding objects to an interface

The following objects, belonging to an interface in the model, can be added to another interface:

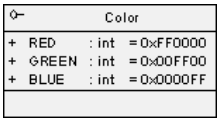
Object	Description
Attribute	Defines a property of the interface
Operation	Implementation of a service that can be requested from any object of the interface in order to affect behavior

When you add an object to an interface, you create a copy of the object. The new object exists as a unique object, and you can then make changes to it as you would to any object in the model. You can also attach inner classifiers to an interface.

For more information on how to attach inner classifiers to an interface, see section Defining inner classifiers.

Adding an attribute to an interface

An attribute is a property of an interface. For example, consider an interface named Color with three attributes RED, GREEN, and BLUE. They are all static, final and frozen.



If you generate in Java, you see:

```
public interface Color
{
   public static final int RED = 0xFF0000;
   public static final int GREEN = 0x00FF00;
   public static final int BLUE = 0x0000FF;
}
```

All these attributes are constants because they are static (independent from the instances), final (they can not be overloaded), and frozen (their value cannot be changed).

You can use the attributes of other interfaces or classes and add them to the current interface.

❖ **To add an attribute to an interface:**

1 Double-click an interface in the diagram to display the interface property sheet.

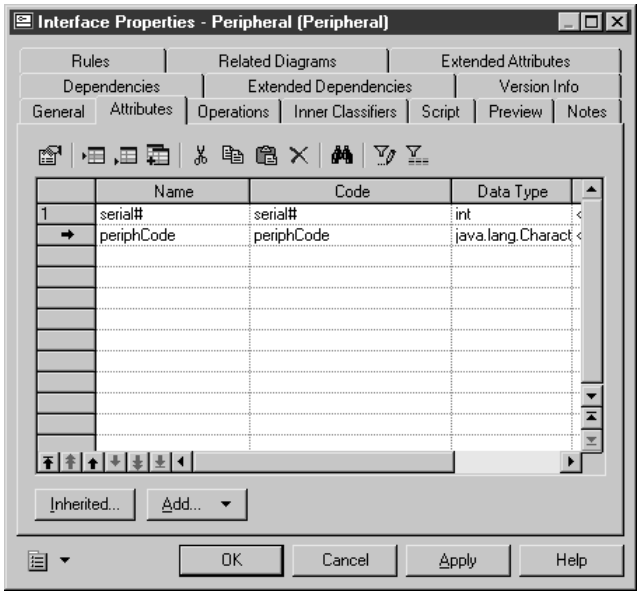
2 Click the Attributes tab to display the Attributes page.

3 Click the Add Attributes tool.

The Selection window appears. It contains a list of all the attributes that exist in the model, except those that already belong to the interface.

4 Select the attributes you want to add to the interface.

5 Click OK.



The attributes are added to the interface and appear at the end of the list.

6 Click OK.

Adding an operation to an interface

An operation is the implementation of a service that can be requested from any object of the class in order to affect behavior.

You can use the operations of other interfaces or classes and add them to the current interface.

❖ To add an operation to an interface:

1 Double-click an interface in the diagram to display the interface property sheet.

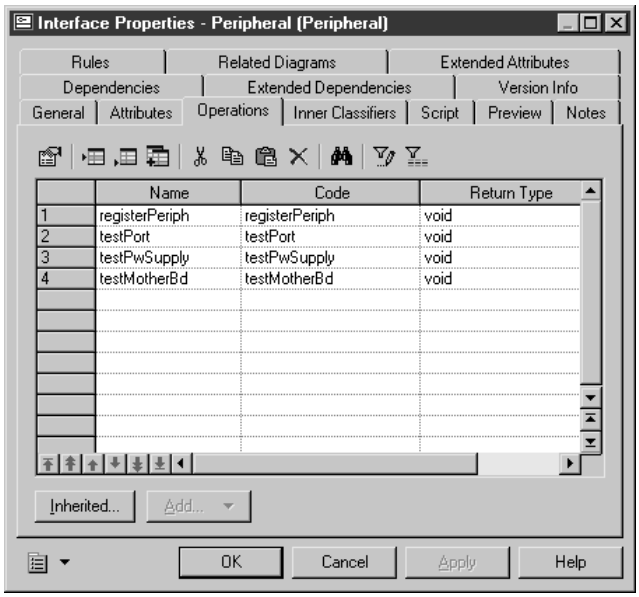
2 Click the Operations tab to display the Operations page.

3 Click the Add Operations tool.

The Selection window appears. It contains a list of all the operations that exist in the model, except those that already belong to the interface.

4 Select the operations you want to add to the interface.

5 Click OK.



The operations are added to the interface and appear at the end of the list.

6 Click OK.

Modifying interface properties

You can modify the interface properties using the following methods:

- From the interface property sheet
- From the list of interfaces

For more information on the different ways to modify the interface properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying interface display preferences

You can modify the following display preferences for interfaces using the Tools→Display Preferences command:

Preference	Description
Show attributes	Displays all the interface attributes, or limits the number displayed to a maximum that you specify in the Limit box
Show operations	Displays all the interface operations, or limits the number displayed to a maximum that you specify in the Limit box
Show stereotype	Displays the stereotype of the interface
Show constraints	Displays the constraints (business rule) attached to the interface
Show inner classifiers	Displays inner classifiers attached to the interface in the model

Defining attributes in a class diagram

An attribute is a named property of a class (or an interface) describing its characteristics.

A class or an interface may have none or several attributes. Each object in a class has the same attributes, but the values of the attributes may be different.

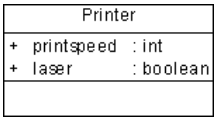
Attribute names within a class must be unique. You can give identical names to two or more attributes only if they exist in different classes.

You can define attributes for the following elements:

- Class
- Interface
- Identifier

Example

The class Printer, contains two attributes: printspeed and laser:



Attribute properties in a class diagram

You access attribute properties from a class or an interface property sheet.

An attribute has the following properties:

Property	Description	Maximum length
Parent	To which the attribute belongs	254
Name	Name of the attribute	254
Code	Reference name of the attribute	254
Comment	Descriptive comment for the attribute	—
Stereotype	Extends the semantics of an attribute derived from existing attributes but specific to your needs	254
Data type	Set of instances sharing the same operations, abstract attributes, relationships, and semantics	—

Property	Description	Maximum length
Visibility	Visibility of the attribute, whose value denotes how it is seen outside its enclosing namespace. Exists only in classes	—
Multiplicity	Specification of the range of allowable number of values the attribute may hold	—
Derived	Indicates that the attribute is a calculated formula. Exists only in classes	—
Static	The attribute cannot be modified. Exists only in classes	—
Initial value	Initial value of the attribute (including quotes if needed)	—
Changeability	Specifies if the value of the attribute can be modified once the object has been initialized	—
Domain	Name of the associated domain	254
Primary Identifier	If selected, indicates that the current attribute is part of a primary identifier. Primary identifiers are converted to primary keys after generation of an OOM to a PDM. Exists only in classes	—
Persistent	Groupbox for valid generation of CDM or PDM persistent models. Exists only in classes	—
Migrated from	Contains the association name that is at the origin of creation of the attribute. You can click the Properties tool beside this box to display the association property sheet	—

For more information on adding operations to a class, see section Adding Getter and Setter operations to a classifier.

The Inherited button in the Attributes page of a class or interface property sheet allows you to view the attributes inherited from the parent classes or interfaces.

An attribute also includes business rules in its properties. You can add business rules in the Rules page.

Attribute visibility in a class diagram

Property	Visible
Private	Only to the class to which it belongs
Protected	Only to the class and its derived objects
Package	To all objects contained within the same package
Public	To all objects

Attribute multiplicity in a class diagram

The cardinality of each attribute is called the multiplicity.

Multiplicity	Number of instances
0..1	Zero or one
0..*	None to unlimited
1..1	Exactly one
1..*	One to unlimited
*	None to unlimited

You can change the default format of multiplicity from the registry.

```
HKEY_CURRENT_USER\Software\Sybase\PowerDesigner
9\ModelOptions\Cld
MultiplicityNotation = 1 (0..1) or 2 (0,1)
```

Attribute changeability in a class diagram

The changeability of an attribute has the following values:

Changeability	Value
Changeable	The value can be changed
Read-only	Prevents the creation of a setter operation (a setter is created in the method inside the class)
Frozen	Constant
Add-only	Allows you to add a new value only

Persistent attribute in a class diagram

The Detail page contains a Persistent groupbox whose purpose is to improve the generation of codes and data types during generation of a conceptual data model or a physical data model from an object-oriented model.

The Persistent groupbox contains the following values:

- Code: code of the entity attribute or column that will be generated in a persistent CDM or PDM model. You have to type the code that you want to give to the attribute or the column in the Code box.
- Data type: specifies a persistent data type used in the generation of a persistent model, either CDM or PDM. The persistent data type is defined from default PowerDesigner conceptual data types
- Length: maximum number of characters of the persistent data type
- Precision: number of places after the decimal point, for persistent data type values that can take a decimal point
- Class generation: when an attribute uses a class as a data type (complex data type) this dropdown listbox allows to define the type of generation for the class used as data type. The dropdown listbox contains the following values:

  <Undefined>
  Persistent
  Embedded

  For more information on class generation, see section Managing complex data type persistence, in chapter Generating from an Object-Oriented Model.

For more information on object persistence, see section Managing object persistence during OOM to PDM generation, in chapter Generating from an Object-Oriented Model.

Persistence of classes attributes
The Persistent groupbox is only available for class attributes, it is not applicable to interface attributes.

Migrated from

The Detail page contains a Migrated from box. It is a read-only value showing the name of the association which roles have been migrated to attributes. The Migrated box only contains a value when association roles have been migrated.

For more information on migrating attributes, see section Migrating association roles in a class diagram.

Creating an attribute in a class diagram

You create an attribute from the class or interface property sheet. To display the class or interface property sheet, you can:

- Double-click the class or the interface symbol
- Right-click the class or the interface node in the Browser and select Properties

At creation, an attribute has a default name including a number, this number is assigned in the order of creation.

❖ **To create an attribute in a class diagram:**

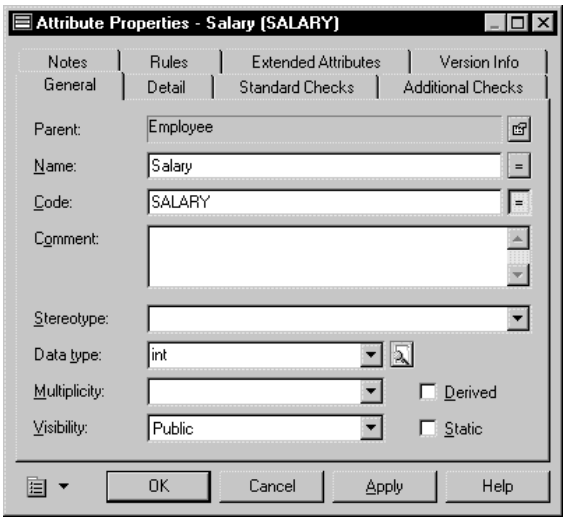
1. Click the Attributes tab in the class or interface property sheet to display the Attributes page.
2. Click a blank line in the list.

   An arrow appears at the beginning of the line.
3. Double-click the arrow at the beginning of the line.

   A confirmation box asks you to commit the object creation.
4. Click Yes.

   The attribute property sheet appears.

5 Type a name and a code.



6 Click OK.

A new attribute is created for the class or the interface.

Attaching an attribute to a domain in a class diagram

If you attach an attribute to a domain, the domain supplies the data type and related data characteristics. It may also indicate check parameters, and business rules.

❖ **To attach an attribute to a domain in a class diagram:**

1 Double-click a class in the diagram to display the class property sheet.

2 Click the Attributes tab to display the Attributes page.

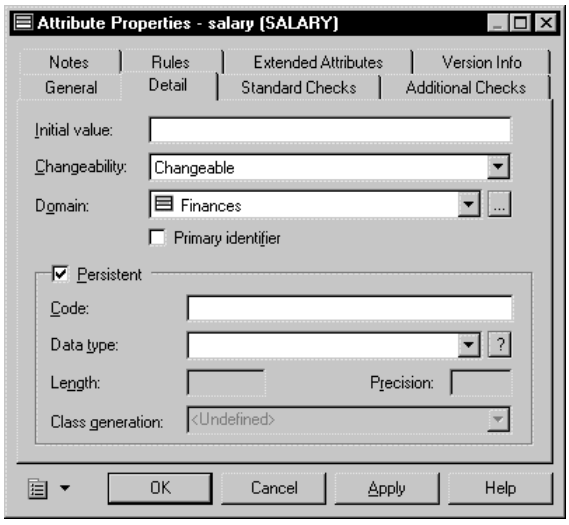
3 Select an attribute in the list.

An arrow appears at the beginning of the line.

4 Click the Properties tool.
or
Double-click the arrow at the beginning of the line.

The attribute property sheet appears.

5 Click the Detail tab.



6 Select a domain from the Domain dropdown listbox at the bottom of the dialog box.
or
Click the Ellipsis button and create a new domain in the List of Domains.

7 Click OK.

You return to the Attributes page. In the Data type column, the domain's data type replaces the data type previously defined for the attribute.

8 Click OK.

Adding Getter and Setter operations to a classifier

Getter or Setter operations are special types of operations you create for an attribute. You create them for sending and receiving data values between attributes of a class or of an interface.

You create a Getter or a Setter operation from the list of attributes. For each attribute, you can create a Getter, a Setter, or both a Getter and Setter operations.

Operation	Description
Getter	Returns a value from an attribute
Setter	Sets a value into an attribute

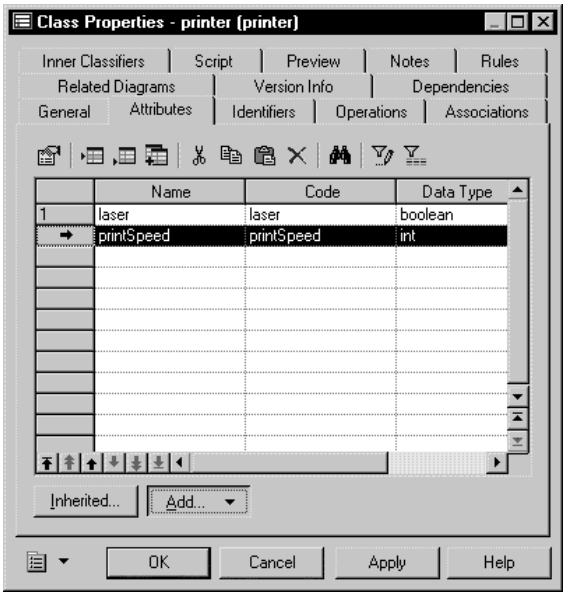
❖ **To add a Getter and a Setter operation to a classifier from an attribute:**

1 Double-click a class or interface in the diagram to display the property sheet.

2 Click the Attributes tab to display the Attributes page.

3 Select one or more attributes.

4 Click the Add button at the bottom of the attributes page.



5 Select Get/Set Operations from the dropdown listbox.

The operation is created for the attributes.

6 Click the Operations tab to display the Operations page.

The newly created operations appear at the bottom of the list of operations. The values are grayed indicating that the names cannot be modified.

7 Click OK.

Copying an attribute to another class

You can duplicate an attribute from one class to another. If the class already contains an attribute with the same name or code as the copied attribute, the copied attribute is renamed. For example the attribute PERIPHLD is renamed into PERIPHLD2 when it is copied to a class which already contains an attribute PERIPHLD.

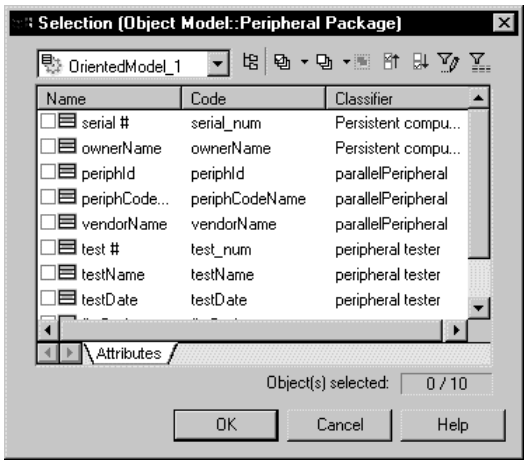
❖ **To copy an attribute to another class:**

1 Double-click a class in the diagram to display the class property sheet.

2 Click the Attributes tab to display the Attributes page.

3 Click the Add Attributes tool.

A selection box lists attributes defined for all the classes in the model.



4 Select one or more attributes in the list.

5 Click OK.

The copied attributes appear in the list of attributes for the current class.

6 Click OK.

Modifying attribute properties in a class diagram

You can modify the attribute properties using the following methods:

- From the attribute property sheet
- From the list of attributes

For more information on the different ways to modify the attribute properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying attribute display preferences in a class diagram

You can modify the following display preferences for attributes using the Tools→Display Preferences command:

Preference	Description
Show visibility	Displays the visibility of the attribute as icons, markers, or keywords
Show stereotype	Displays the stereotype of the attribute
Show data type	Displays the attribute data type in the class or interface symbol. You can choose to display the data type or the domain attached to the attribute
Show domain	Displays the domain attached to the attribute
Show initial value	Displays the initial value of the attribute upon instantiation of the class

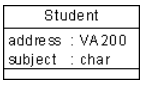
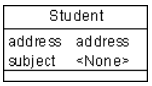
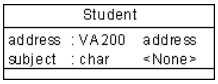
Visibility

You can modify the aspect of the visibility property of an attribute:

Visibility	When selected
Use icons	Displays the attribute as an icon
Use markers	Displays the visibility of the attribute as a marker: - (private), # (protected), + (public), or * (package)
Use keywords	Displays the visibility of the attribute as a word: private, protected, public, or package

Display domain and data type

The display options for domain and data types are the following:

Selected check box	Result
Show data type	
Show domain	
Show data type Show domain	

Defining identifiers

An identifier is a class attribute, or a combination of class attributes, whose values uniquely identify each occurrence of the class. It is used during inter-model generation when you generate a CDM or a PDM into an OOM, the CDM identifier and the PDM primary or alternate keys become identifiers in the OOM.

Each class can have at least one identifier. Among identifiers, the primary identifier is the main identifier of the class. This identifier corresponds to a primary key in the PDM.

When you create an identifier, you can attach attributes or business rules to it. You can also define one or several attributes as being primary identifier of the class.

Identifier example

The social security number for a class employee is the primary identifier of this class.

Identifier properties

You can access the identifier properties from a class property sheet.

An identifier has the following properties:

Property	Description	Maximum length
Name	Name of the identifier	254
Code	Reference name of the identifier	254
Comment	Descriptive comment for the identifier	—
Class	Name of the class to which the identifier belongs	254
Primary identifier	Indicates that it is the primary identifier of the class. There can only be one primary identifier per class	—

An identifier also includes the following properties:

Property	Description
Attributes	Define the characteristics of an identifier
Rules	Business rules attached to the identifier

Creating a primary identifier

There are several ways to create a primary identifier:

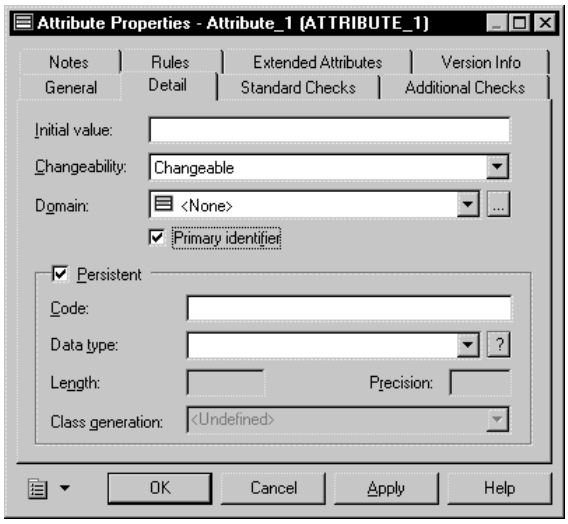
- Create a primary identifier while you create the class attributes
- Define the primary identifier from the list of identifiers

At creation, an identifier has a default name including a number, this number is assigned in the order of creation.

Creating a primary identifier while you create the class attributes

❖ **To create a primary identifier while you create the class attributes:**

1. Double-click a class in the diagram to display the class property sheet.
2. Click the Attributes tab to display the Attributes page.
3. Double-click an attribute in the list to display the property sheet of the attribute.
4. Click the Detail tab.
5. Select the Primary Identifier check box.



6. Click OK.

   You return to the Attributes page of the class property sheet.

7. Click the Identifiers tab to display the Identifiers page.

A new identifier appears in the list.

8 Double-click the identifier in the list to display the property sheet of the identifier.

9 Click OK in each of the dialog boxes.

Defining the primary identifier from the list of identifiers

❖ **To define the primary identifier from the list of identifiers:**

1 Select Model→Identifiers to display the list of identifiers.

2 Double-click an identifier in the list to display the property sheet of the identifier.

3 Select the Primary Identifier check box.

4 Click OK in each of the dialog boxes.

Creating an identifier

You create an identifier from the class property sheet. To display the class property sheet, you can:

- Double-click the class symbol
- Right-click the class node in the Browser and select Properties

❖ **To create an identifier:**

1 Click the Identifiers tab in the class property sheet.

2 Click the Add a row or the Insert a row tool.

A new identifier appears in the list.

3 Type a name and a code.

4 Click OK.

Adding attributes to an identifier

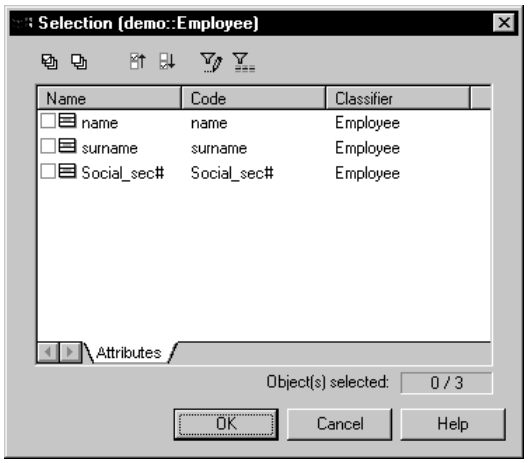
An identifier can contain one or several attributes. You can add these attributes to an identifier to further characterize the identifier.

❖ **To add attributes to an identifier:**

1 From the Identifiers page, click the Properties tool.

2 Click the Attributes tab to display the Attributes page for the identifier.

3 Click the Add Attributes tool to display the list of attributes for the class.



4 Select check boxes for attributes you want to add to the identifier.

5 Click OK in each of the dialog boxes.

Modifying identifier properties

You can modify the identifier properties using the following methods:

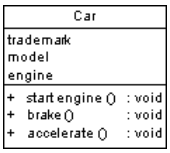
- From the identifier property sheet
- From the list of identifiers

For more information on the different ways to modify the identifier properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Defining operations

An operation is a named specification of a service that can be requested from any of a class's objects to affect behavior. It is a specification of a query that an object may be called to execute. A class may have any number of operations or no operations at all.

In the following example, the class Car, has 3 operations: start engine, brake, and accelerate.



Operations have a name and a list of parameters. Operation names within a class must be unique. You can give identical names to two or more operations only if they exist in different classes.

For more information on **EJB** operations, see section Defining operations for EJBs, in chapter Working with platform dependent components.

Operation properties

An operation has the following properties:

Property	Description	Maximum length
Parent	To which the operation belongs	254
Name	Name of the operation, must be unique within the class or interface	254
Code	Reference name of the operation	254
Comment	Descriptive comment for the operation	—
Stereotype	Extends the semantics of an operation derived from existing operations but specific to your needs	254
Return Type	A list of values returned by a call of the operation. If none are returned, the return type value is null	—
Visibility	Visibility of the operation, whose value denotes how it is seen outside its enclosing name space. Exists only in classes	—

Property	Description	Maximum length
Language event	When classes represent elements of interfaces, this box allows you to show an operation as triggered by a significant occurrence of an event	—
Static	The operation cannot be modified	—
Array	Flag defining the return type of the operation. It is true if the value returned is a table	—
Abstract	The operation cannot be instantiated and thus has no direct instances	—
Final	The operation cannot be redefined	—
Read-only	Operation whose execution does not change the class instance	—
Web service method	If displayed and selected, implies that the operation is used as a web service method	—

For more information on adding operations using the Add button, see section Adding operations to a class.

For more information on inherited operations, see section Adding an operation from a parent class.

For more information on implemented operations, see section Creating an implementation operation.

Operation stereotype

Property	Description
<<constructor>>	Operation called during the instantiation of an objet that creates an instance of a class
<<create>>	Operation used by a class when instantiating an object
<<destroy>>	Operation used by a class that destroys an instance of a class
<<storedProcedure>>	Operation will become a stored procedure in the generated PDM
<<storedFunction>>	Operation will become a stored function in the generated PDM
<<EJBCreateMethod>>	EJB specific CreateMethod

Property	Description
<<EJBFinderMethod>>	EJB specific FinderMethod
<<EJBSelectMethod>>	EJB specific SelectMethod

For more information on EJB specific methods, see section Defining EJBs, in chapter Working with platform dependent components.

Operation visibility

Property	Visible
Private	Only to the class to which it belongs
Protected	Only to the class and its derived objects
Package	To all objects contained within the same package
Public	To all objects

Operation related diagrams

In the property sheet of an operation, you can select the Related Diagrams tab to display all diagrams of the model that are related to an operation. They can be attached and detached from the current operation at this stage. This page displays a list of the diagrams identified by their name and code.

For more information on related diagrams, see section Defining related diagrams, in chapter Working with Object-Oriented Models.

Defining operation parameters

A parameter is a specification of a variable that can be changed, passed, or returned. Parameters are used for operations in a class diagram.

A parameter always has a type indicating the flow of information, a name and a code.

Operation parameter properties

A parameter has the following properties:

Property	Description	Maximum length
Parent	Operation to which the parameter belongs	254
Name	Name of the parameter	254
Code	Reference name of the parameter	254
Comment	Descriptive comment for the parameter	—
Data type	Set of instances sharing the same operations, abstract attributes, relationships, and semantics	—
Array	When selected, turns attributes into table format (for generation and reverse engineering of Java and PowerBuilder)	—
Parameter Type	Direction of information flow for the parameter	—
Default value	Default value when a parameter is omitted	—
WSDL data type	Only available with Web services. Defines the XML Schema/SOAP type used during invocation of a web method (using http or Soap)	—

Default value

Here is an example to describe the Default value property:

Use an operation `oper(string param1, integer param2)`, and specify two arguments `oper(val1, val2)` during invocation. Some languages, like C++, allow you to define a default value that is then memorized when the parameter is omitted during invocation.

If the declaration of the method is `oper(string param1, integer param2 = default)`, then the invocation `oper(val1)` is similar to `oper(val1, default)`.

Parameter type

The value in the Parameter type dropdown listbox indicates what is returned when the parameter is called by the operation during the execution process.

You can set the following values for the direction:

Value	Description
In	Input parameter passed by value. The final value may not be modified and information is not available to the caller

Value	Description
In/Out	Input parameter that may be modified. The final value may be modified to communicate information to the caller
Out	Output parameter. The final value may be modified to communicate information to the caller

Creating an operation parameter

You create a parameter from the operation property sheet. To display the operation property sheet, you can:

- Double-click the class or interface symbol, click the Operations tab and double-click the arrow at the beginning of the line
- Right-click the operation node in the Browser and select Properties

❖ To create an operation parameter:

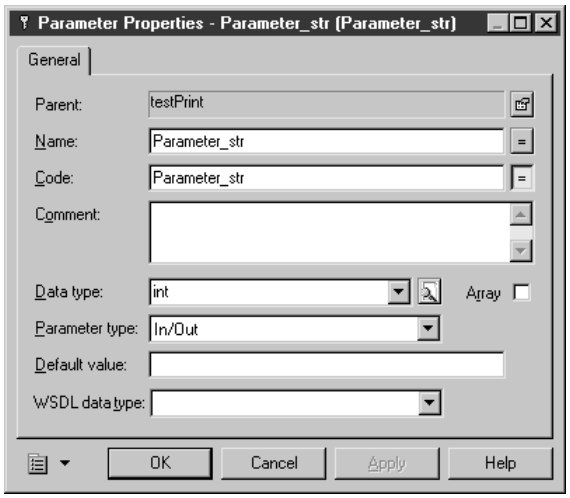
1 Select the Parameters tab in the operation property sheet.

2 Click the Add a Row tool
or
Select a row.

An arrow appears at the beginning of the line.

3 Double-click the arrow at the beginning of the line to display the parameter property sheet.

4 Type a name and code.



5 Select a value from the Parameter Type dropdown listbox.

6 Click OK.

The parameter is added to the operation.

Modifying operation parameter properties

You can modify operation parameter properties from the parameter property sheet.

❖ **To modify an operation parameter from the property sheet:**

1 Double-click a class in the diagram to display the class property sheet.

2 Click the Operations tab to display the operation page.

3 Double-click the arrow at the beginning of the line to display the operation property sheet.

4 Click the Parameters tab to display the Parameters page.

5 Select the parameter you want to modify and click the Properties tool to display the parameter property sheet.

6 Modify parameter properties as required.

7 Click OK in each of the dialog boxes.

Creating an operation

You create an operation from the class or interface property sheet. To display the operation property sheet, you can:

- Double-click the class or interface symbol
- Right-click the class or interface node in the Browser and select Properties

At creation, an operation has a default name including a number, this number is assigned in the order of creation.

❖ **To create an operation:**

1 Click the Operations tab in the class or interface property sheet to display the Operations page.

2 Click a blank line in the list.

An arrow appears at the beginning of the line.

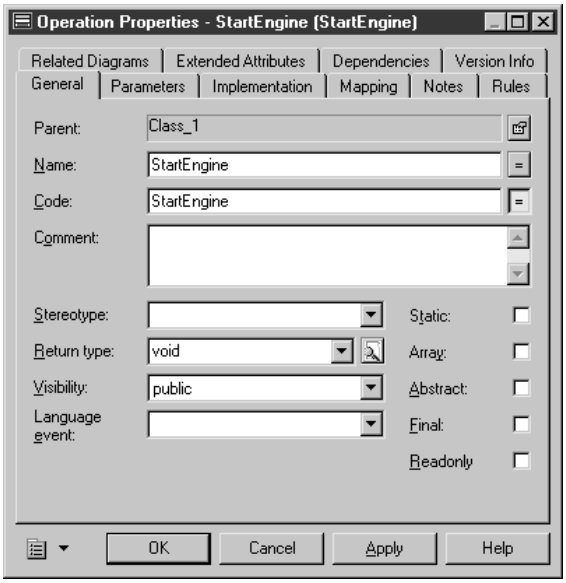
3 Double-click the arrow at the beginning of the line.

A confirmation box asks you to commit the object creation.

4 Click Yes.

The operation property sheet appears.

5 Type a name and a code.



For opening property sheets at last accessed page, see the framed text in Attaching an inner classifier to a class or an interface.

6 Click OK.

The operation is added to the class or the interface.

Adding operations to a class

You can add the following operations to a class.

For more information on adding EJB operations, see section Defining operations for EJBs, in chapter Working with platform dependent components.

Adding a duplicate operation to a class

A duplicate operation is an operation that creates and initializes an instance of a class within the class.

When you create a duplicate operation, it has the name Duplicate which you can modify. You can define only one duplicate operation to any given class.

❖ **To add a duplicate operation to a class:**

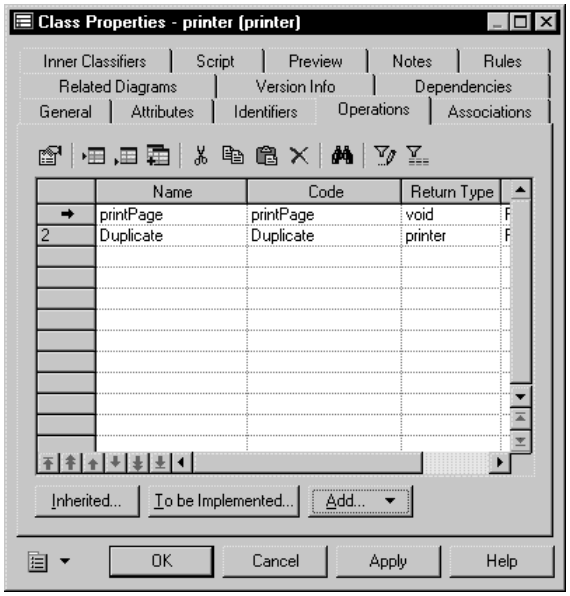
1 Double-click a class in the diagram to display the class property sheet.

2 Click the Operations tab to display the Operations page.

3 Click the Add button at the bottom of the page.

4 Select Duplicate Operation from the dropdown listbox.

A duplicate operation, with the name Duplicate, is created at the end of the list of operations.



5 Click OK.

Adding an operation from a parent class

Within the context of an inheritance, you can add to a class an operation that belongs to a parent class. The new operation has the same signature (name and parameters) as the original operation, but does not share other properties.

Once you add an operation to a class, you can only modify the code implementation of the operation. You cannot modify the signature of the operation.

You can click the Inherited button in the Operations page of a class or interface property sheet to view the operations inherited from the parent classes or interfaces.

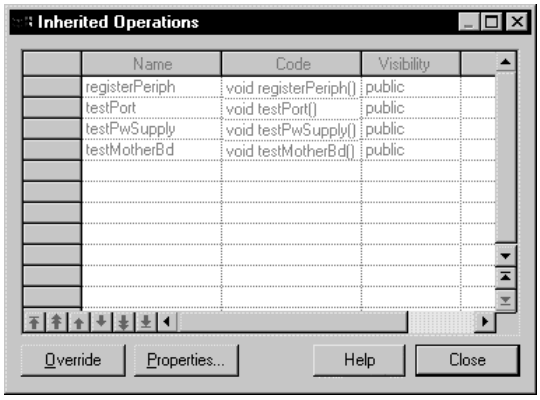
❖ **To add an inherited operation to a class:**

1 Double-click a class that is linked to a parent class in the diagram.

2 Click the Operations tab to display the Operations page.

3 Click the Inherited button.

The Inherited Operations window appears with the operations that belong to all the parent classes of the class.



4 Select an operation.

5 Click Override and then Close.

A copy of the operation is added to the list of operations. It is grayed indicating that its properties cannot be modified. The Override button is used to totally override an existing operation.

6 Click OK in each of the dialog boxes.

Adding Default constructor/destructor to a class

A constructor is a special type of operation that creates and initializes an instance of a class. A destructor is the complement of a constructor, it deinitializes and destroys the class instance. You create constructors and destructors from the Operations page of the class property sheet.

If the current object language of the OOM is PowerBuilder, you can create one Default constructor and one Default destructor for a class. Default means that it has no parameters. The constructor is named "constructor" and the destructor is named "destructor", you cannot modify these names.

If you are using another object language, you can create only one Default constructor and no Default destructor for a class. The constructor has the same name as the class to which it belongs.

You cannot declare a Return Type for a constructor.

❖ **To add a Default constructor and destructor to a class:**

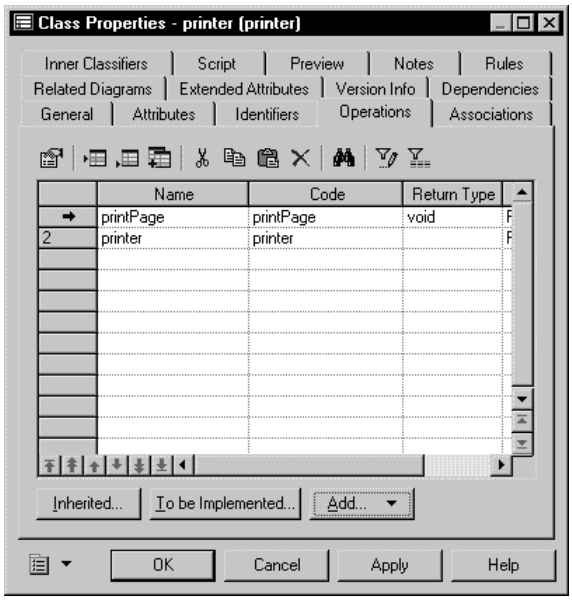
1 Double-click a class in the diagram to display the class property sheet.

2 Click the Operations tab to display the Operations page.

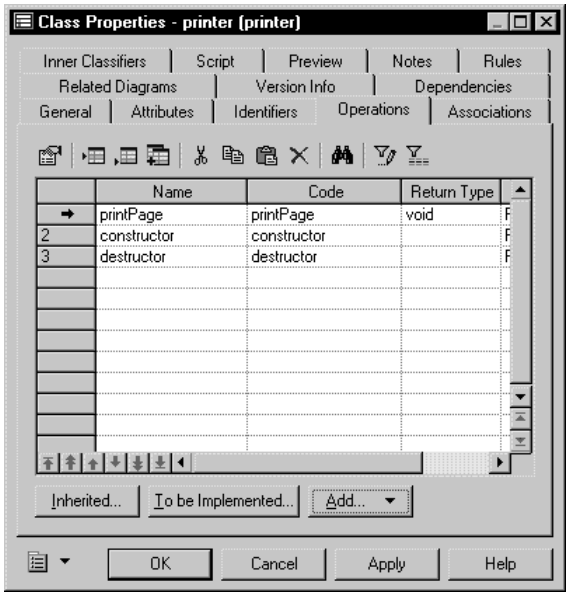
3 Click the Add button at the bottom of the page.

4 Select Default Constructor/Destructor from the dropdown listbox.

If the current object language is not PowerBuilder, a Default constructor is created at the end of the list of operations. It has the same name as the class to which it belongs:



If the current object language of the OOM is PowerBuilder, a Default constructor and a Default destructor are created at the end of the list of operations. They have the names constructor and destructor:



5 Click OK.

Adding a Copy constructor to a class

A constructor is a special type of operation that creates and initializes an instance of a class. A Copy constructor contains a copy of the attributes of the class that exist at the moment of creation of the constructor.

When you create a Copy constructor, it has the same name as the class. It has a default parameter, prefixed by the keyword *old*, which you can modify, or you can add other parameters.

If the class is a parent of another class, the attribute names within the body of the Copy constructor operation are the same as those in the parent class.

You can define only one Copy constructor for any given class.

❖ **To add a Copy constructor to a class:**

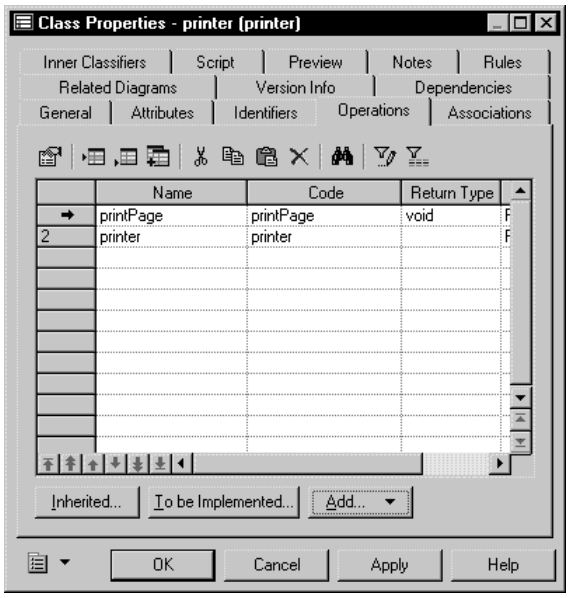
1 Double-click a class in the diagram to display the class property sheet.

2 Click the Operations tab to display the Operations page.

3 Click the Add button at the bottom of the page.

4 Select Copy Constructor from the dropdown listbox.

A Copy constructor is created at the end of the list of operations for the class. It has the same name as the class to which it belongs.



5 Click OK.

Adding activate/deactivate operations to a class

Activate and deactivate operations can be generated for PowerBuilder objects. This feature is only available if the object language of your model is PowerBuilder.

❖ **To add an activate and deactivate operation to a class:**

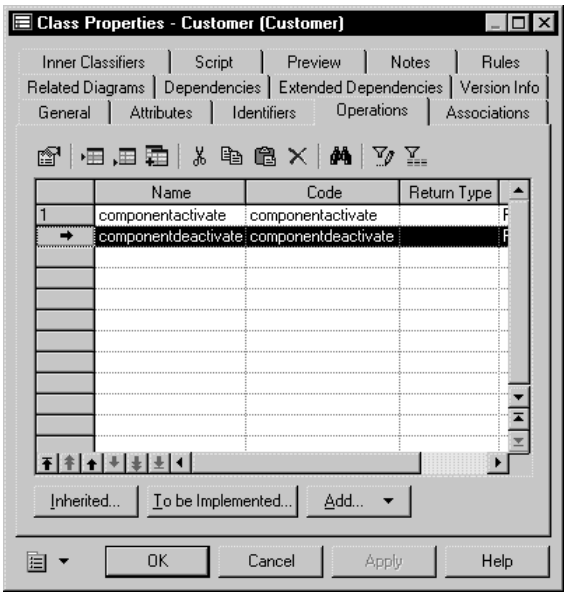
1 Double-click a class in the diagram to display the class property sheet.

2 Click the Operations tab to display the Operations page.

3 Click the Add button at the bottom of the page.

4 Select Activate/Deactivate Operations from the dropdown listbox.

The componentactivate and componentdeactivate operations are automatically created.



5 Click OK.

Adding a static initializer

The static initializer is a block of code whose purpose is class initialization. It is executed before the class constructor and is handled as a specific method prefixed with the name: _STATIC_INITIALIZER.

It is only available in Java. The static initializer method's name and code are read-only and its parameters are ignored.

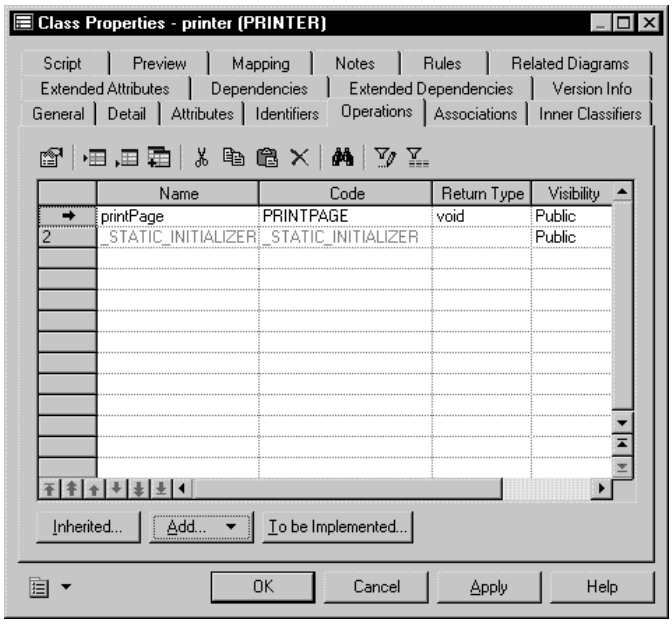
❖ **To add a static initializer:**

1 Double-click a class in the diagram to display the class property sheet.

2 Click the Operations tab to display the Operations page.

3 Click the Add button at the bottom of the page.

4 Select Static Initializer from the dropdown listbox.



The _STATIC_INITIALIZER value appears at the bottom of the list of operations. The value is grayed indicating that the name cannot be modified.

5 Click OK.

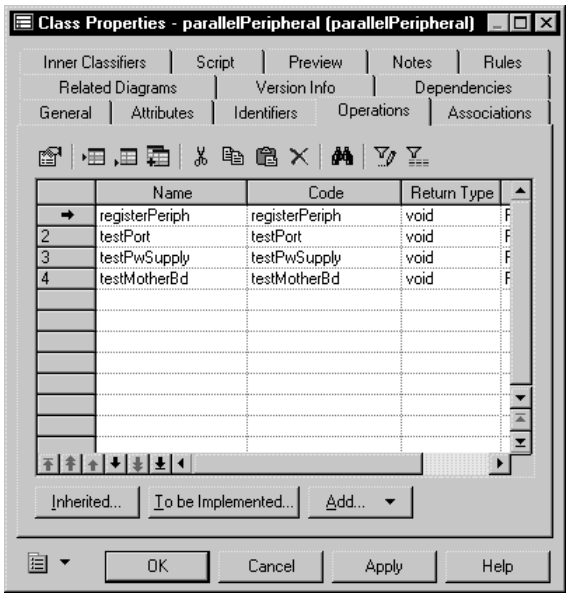
Creating an implementation operation

If you create a realization link between a class and an interface in which the class implements the interface, you create an operation in the class that implements the interface.

❖ **To create an implementation operation:**

1 Double-click a class that is linked to an interface by a realization link.

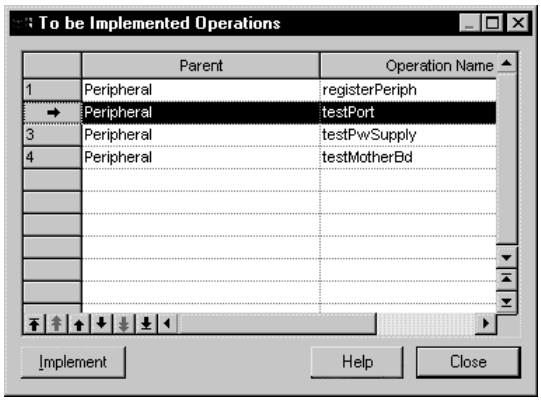
2 Click the Operations tab to display the Operations page.



3 Click the To be Implemented button.

The To Be Implemented Operations window appears. It contains a list of all operations of the interface that can be implemented from the class.

4 Select an operation from the list.



5 Click Implement and then Close.

A copy of the operation is created in the class and is added to the list of operations. It is grayed, indicating that its name cannot be modified.

6 Click OK.

Modifying the code of an implementation operation

You modify the code of an implementation operation from the Implementation page of the operation property sheet.

The Implementation page contains the following tabbed pages:

Items	Description
Body	Code of the implementation
Exceptions	Signal raised in response to behavioral faults during system execution
Pre-condition	A constraint that must be true when an operation is invoked
Post-condition	A constraint that must be true at the completion of an operation
Specification	Similar to the pseudo code, it is a description of the normal sequence of actions

You can use the following tool from the Exceptions page in the Implementation page toolbar:

Tool	Description
	Add Exception tool. Opens a selection window that allows you to select an exception classifier to be added. The exception classifier is added where the cursor is positioned

❖ **To modify the code of an implementation operation:**

1 Double-click an implementation operation in the Operations page of a class that implements an interface.

The operation property sheet appears.

2 Click the Implementation tab to display the Implementation page.



3 Click a tab at the bottom of the edit window and type or modify code.

4 Click OK in each of the dialog boxes.

Copying an operation to another class

You can copy an operation from one class and add it to another. If the class already contains an operation with the same name or code as the copied operation, the copied operation is renamed. For example the operation testPort is renamed testPort2 when it is copied to a class which already contains an operation testPort.

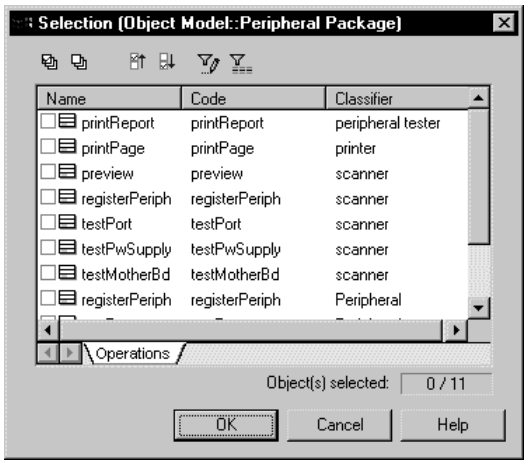
❖ **To copy an operation to another class:**

1 Double-click a class in the diagram to display the class property sheet.

2 Click the Operations tab to display the Operations page.

3 Click the Add Operations tool.

A selection box lists operations attached to all the classes in the model.



4 Select one or more operations in the list.

5 Click OK.

The copied operations appear at the end of the list of operations.

6 Click OK.

Modifying operation properties

You can modify the operation properties using the following methods:

- From the operation property sheet
- From the list of operations

For more information on the different ways to modify the operation properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying operation display preferences

You can modify the following display preferences for operations using the Tools→Display Preferences command:

Preference	Description
Show visibility	Displays the operation as icons, or markers, or keywords
Show stereotype	Displays the stereotype of the operation
Show return type	Displays the operation return type
Show parameters	Displays the operation parameters

The visibility of an operation in a class or interface can be displayed as:

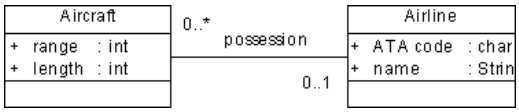
Visibility	When selected
Icon	Displays the operation as an icon
Use Markers	Displays the visibility of the operation as a marker: - (private), # (protected), + (public), or * (package)
Use Keywords	Displays the visibility of the operation as a word: private, protected, public, or package

Defining associations in a class diagram

An association represents a structural relationship between classes or between a class and an interface. It is drawn as a solid line between pairs of classes, or classes and interfaces.

You can define an association between two classes, or between a class and an interface.

Example



An association must have a name but the name you give to the association is less important if you use association roles.

Association Roles

Each side of an association may have a name called a role, the role describes the function of a class as viewed by the opposite class.

Association properties in a class diagram

You can double-click any association symbol in a diagram to display its properties:

Property	Description	Maximum length
Name	Name of the association. Calculated, read-only value	254
Code	Reference code of the association. Calculated, read-only value	254
Comment	Descriptive comment for the association	—
Stereotype	Extends the semantics of an association derived from existing associations but specific to your needs. <<implicit>> means that the relationship is not manifest, it is only conceptual	254
Role A	One side of an association. Each role can have a name and a cardinality and be navigable	254
Class A	Name of the class at one end of the association	254
Role B	One side of an association. Each role can have a name and a cardinality and be navigable	254
Class B	Name of the class at the other end of the association	254

Property	Description	Maximum length
Association Class	Class related to the current association that completes the association definition	254

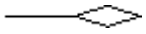
Association role in a class diagram

Each end of an association is called a **role**. You can define its multiplicity and define one of the two roles as an aggregation or a composition. You can also define a visibility for the association.

Property	Description
Multiplicity	Minimum and maximum number of instances of the association
Ordering	The association is included in the ordering which sorts the list of associations by their order of creation.
Visibility	Visibility of the role of the association whose value denotes how it is seen outside its enclosing namespace
Navigable	Indicates whether or not information can be transmitted between the two objects linked by the relationship
Aggregation	A form of association that specifies a part-whole relationship between a class and an aggregate class (example: a car has an engine and wheels)
Composition	A form of aggregation but with strong ownership and coincident lifetime of parts by the whole; the parts live and die with the whole (example: an invoice and its invoice line)

The Aggregation tool and the Composition tool available from the tool palette create an association, and preset the association properties to make it an aggregation or a composition.

The aggregation symbol in a diagram is the following:



The composition symbol in a diagram is the following:



Association multiplicity in a class diagram

The allowable cardinalities of a role are called multiplicity. Multiplicity indicates the maximum and minimum cardinality that a role can have.

Multiplicity	Number of instances
0..1	Zero or one
0..*	None to unlimited
1..1	Exactly one
1..*	One to unlimited
*	None to unlimited

An extended attribute exists for each role of an association. It allows you to choose how the association should be implemented. They are available in your current object language, from the Profile\Association\ExtendedAttributes category, under the 'roleAContainer' and 'roleBContainer' names. Such extended attributes are pertinent only for a 'many' multiplicity (represented by *), they provide a definition for collections of associations.

For more information on the object language, see section Understanding the object language editor, in chapter Object Languages Reference Guide in the *Advanced User Documentation*.

Association ordering in a class diagram

You can define the ends of an association as being ordered or sorted.

Property	Indicates
Sorted	The set of objects at the end of an association is arranged according to the way they are defined in the model
Ordered	The set of objects at the end of an association is arranged in a specific order
Unordered	The end of an association is neither sorted nor ordered

Association visibility in a class diagram

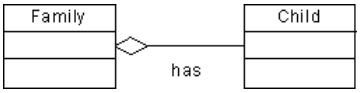
The visibility of an association role is the way it can be seen by other objects. When the role of an association is visible to another object, it may influence the structure or behavior of the object, or similarly, the other object can affect the properties of the association.

Property	Visible
Private	Only to the class to which it is associated
Protected	Only to the class and its derived objects
Package	To all objects contained within the same package
Public	To all objects

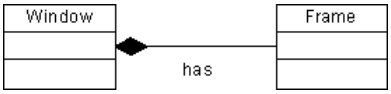
Aggregation/composition

You can define one of the roles of an association as being either an aggregation or a composition in the Aggregation/Composition groupbox.

An **aggregation** is a type of association in which one class represents a larger thing (a whole) made of smaller things (the parts). This special association is a "has-a" link. It means that an object of the whole has objects of the part. In the following example, the family is the whole that can contain children.



A **composition** is also a whole/part type of association in which the parts are strongly tied to the whole. In a composition, an object may be a part of only one composite at a time, and the composite object manages the creation and destruction of its parts. In the following example, the frame is a part of a window. If you destroy the window object, the frame part also disappears.



The following properties also have to be defined:

Property	Description
Container	Specifies which of the two roles is an aggregation or a composition
Indicator	Indicates that the association is an aggregation or a composition

Creating an association in a class diagram

You create an association between two classes or between a class and an interface:

- From a diagram
- From the Browser

At creation, an association has a default name including a number, this number is assigned in the order of creation.

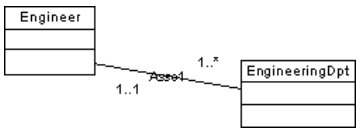
For more information on the different ways to create an association, see section Creating an object, in chapter Managing objects in the *General Features Guide*.

❖ **To create an association from a class diagram:**

1 Click the Association tool.

2 Click inside the first class or interface and while continuing to hold down the mouse button, drag the cursor to the second class or interface. Release the mouse button inside the second class or interface.

The association link appears between both objects.



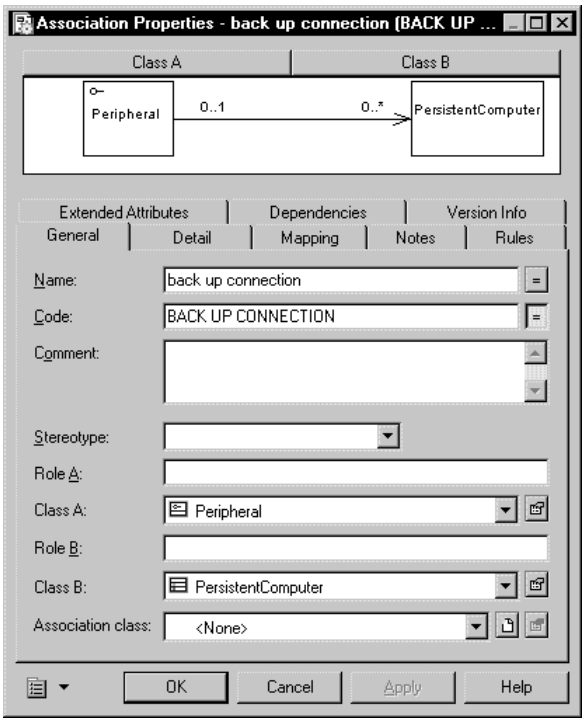
Dragging an association to a different class
You change the class at either end of an association by clicking the association to select it, pressing down CTRL, and dragging one of the attach points to a different class.

3 Click the pointer tool.
Or
Click the right mouse button.

You release the Association tool.

4 Double-click the association in the diagram to display the association property sheet.

5 Type a name and a code.



You can open the property sheet of both classes linked by the association by clicking the Class A and Class B buttons located in the upper part of the association property sheet.

6 Type a name for Role A and Role B if applicable.

7 Click OK.

Creating an association class

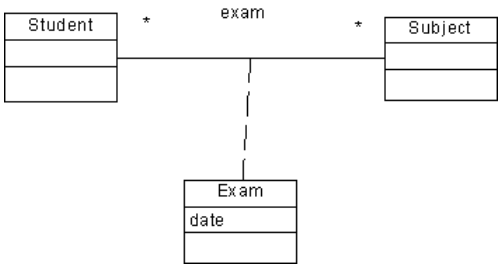
You can add properties to an association between classes or interfaces by creating an **association class**. It is used to further define the properties of an association by adding attributes and operations to the association.

An association class is an association that has class properties, or a class that has association properties. In the diagram, the symbol of an association class is a connection between an association and a class. Association classes must be in the same package as the parent association; you cannot use the shortcut of a class to create an association class.

The class used to create an association class cannot be reused for another association class. However, you can create other types of links to and from this class.

Example

In the following example, the classes Student and Subject are related by an association exam. However, this association does not specify the date of the exam. You can create an association class called Exam that will indicate additional information concerning the association.



❖ **To create an association class:**

1 Create an association between two classes in the model.

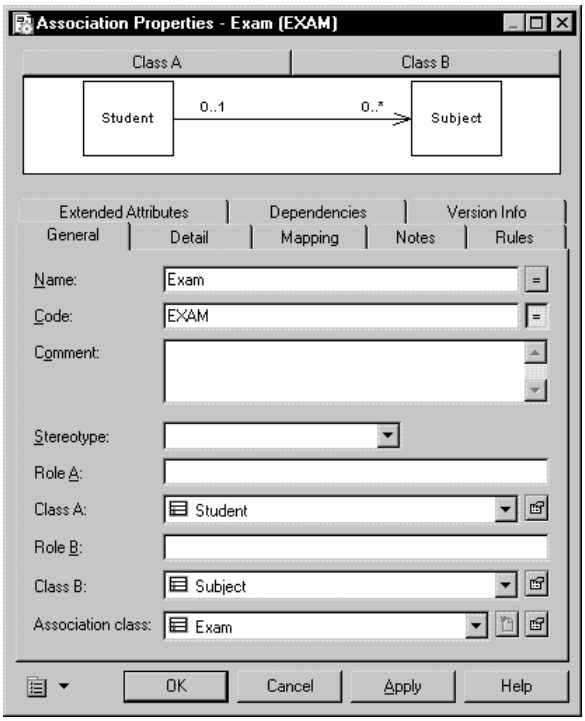
2 Right-click the association to display the association contextual menu.

3 Select Add Association Class.

An association class is created and linked to the association. The association class inherits the name of the original association.

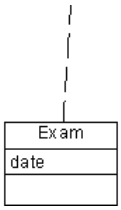
4 Double-click the association link in the diagram to display the association property sheet.

The name of the class that complements the definition of the association is displayed in the Association Class dropdown listbox.



5 Click OK.

A dashed link is automatically added between the class and the association.



Creating an association class from the association property sheet
When you click the Create tool beside the Association Class dropdown listbox, a confirmation box asks you to commit the object creation and a class property sheet appears. You can define the properties of the new class and click OK. The class name appears in the Association Class dropdown listbox and in the diagram.

Migrating association roles in a class diagram

It is possible to migrate association roles and create attributes before generation. This is very convenient for many reasons including data type customization and the ability to change the attribute order in the list of attributes. The last feature is especially important in XML.

Regardless of the navigability, the migration creates an attribute and sets its properties as follows:

- Name and code of the attribute: association role if already set, if not the association name
- Data type: code of the classifier linked by the association
- Multiplicity: role multiplicity
- Visibility: role visibility

Migration rules

The following rules apply when migrating association roles:

- If the migrated attribute name is the same as the role name, then modifying the role name synchronizes the migrated attribute name
- If the migrated attribute data type is the same as the role classifier, then modifying the role multiplicity synchronizes the migrated attribute multiplicity
- If the code of the classifier, linked by the association, changes, then the migrated attribute data type is automatically synchronized
- If you manually change the migrated attribute, the synchronization does not work, the association role is not synchronized
- The migrated attribute is automatically deleted if the association is deleted

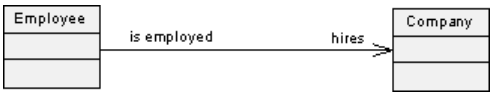
After migration, the property sheet of the new attribute displays the name of the association in the Migrated from box in the Detail page.

Migrating navigable roles

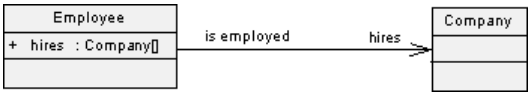
You can migrate the navigable role of an association and turn it into an attribute:

❖ **To migrate navigable roles:**

1 Right-click the association in the diagram.



2 Select Migrate→Migrate Navigable Roles from the association contextual menu.



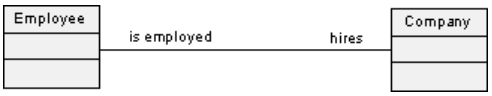
An attribute is created and named after the navigable role of the association followed by the classifier's code.

Migrating both roles

You can migrate both roles of an association and turn them into attributes. An association has two navigable roles either when both navigable check boxes are selected, or when none are selected:

❖ **To migrate both roles:**

1 Right-click the association in the diagram.



2 Select Migrate→Migrate Both Roles from the association contextual menu.



An attribute is created in each class, it is named after each navigable role of the association followed by each classifier's code.

Rebuilding data type links

If a classifier data type is not linked to its original classifier, you can use the Rebuild Data Type Links feature to restore the link. This feature looks for all classifiers of the current model and links them, if needed, to the original classifier.

The Rebuild Data Type Links scans the following data types:

- Attribute data type: an association is created and the attribute is flagged as Migrated Attribute
- Parameter data type: an association is created and links the original classifier
- Operation return type: an association is created and links the original classifier

In some cases, for C++ in particular, this feature is very useful to keep the synchronization of the link even if the data type changes, so that it keeps referencing the original class.

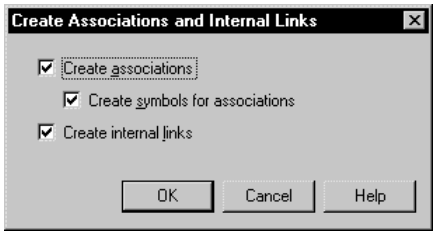
The Rebuild Data Type Links contains the following options:

Option	Description
Create associations	Looks for attributes whose data type matches a classifier and links the attributes to the newly created association as migrated attributes
Create symbols for associations	Creates a symbol of the new association
Create internal links	Creates a link between the return type or parameter data type and the classifier it references

❖ **To rebuild data type links:**

1 Select Tools→Rebuild Data Type Links.

The Create Associations and Internal Links window appears.



2 Select check boxes as required.

3 Click OK.

Modifying association properties in a class diagram

You can modify the association properties using the following methods:

- From the association property sheet
- From the list of associations

For more information on the different ways to modify the association properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying association display preferences in a class diagram

You can modify the following display preferences for associations using the Tools→Display Preferences command:

Preference	Description
Show Multiplicity	Displays the multiplicity of the relationship. You can choose between the actual number of instances (String) or the symbol at each end of the relationship (Symbol)
Show stereotype	Displays the stereotype of the association
Show name	Displays the name of the association
Show constraints	Displays the constraints (business rules) of the association
Show role names	Displays the name of the association roles
Show ordering	Displays the ordering type chosen

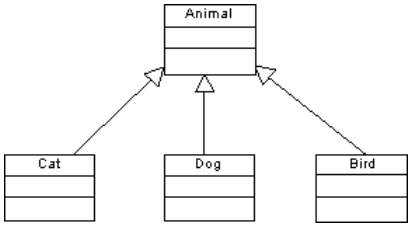
Defining generalizations in a class diagram

A **generalization** is a relationship between a more general element (the parent) and a more specific element (the child). The more specific element is fully consistent with the more general element and contains additional information.

For example, an animal is a more general concept than a cat, a dog or a bird. Inversely, a cat is a more specific concept than an animal.

Example

Animal is a super class. Cat, Dog and Bird are sub-classes of the super class.



Generalizations exist in the following diagrams:

- Class diagram

  A generalization can be created between:

  Two classes

  Two interfaces

  You can also create a generalization between a shortcut of a class and a class, a shortcut of an interface and an interface. Whenever a link is oriented, only the parent object can be the shortcut.

- Use case diagram

  For more information on generalizations in a use case diagram, see section Defining generalizations in a use case diagram, in chapter Building a Use Case Diagram.

- Component diagram

  For more information on generalizations in a component diagram, see section Defining generalizations in a component diagram, in chapter Building a Component Diagram.

Generalization properties in a class diagram

You can double-click the generalization symbol in a diagram to display its properties.

A generalization has the following properties:

Property	Description	Maximum length
Name	Name of the generalization	254
Code	Reference name of the generalization	254
Comment	Descriptive comment for the generalization	—
Parent	Parent class or interface of the generalization	254
Child	Child class or interface of the generalization	254
Stereotype	Extends the semantics of a generalization derived from existing generalizations but specific to your needs. <<implementation>> means that the child object inherits the implementation of the parent object but does not make its interfaces public, nor support them, thus violating its substitutability	254
Visibility	Visibility of the generalization, whose value denotes how it is seen outside its enclosing namespace	—

Visibility

Property	Visible
Private	Only to the child class
Protected	Only to the class and its children
Package	To all objects contained within the same package
Public	To all objects

Creating a generalization in a class diagram

There are several ways to create a generalization:

- From a diagram
- From the Browser

At creation, a generalization has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a generalization, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a generalization from a class diagram:**

1 Click the Generalization tool.

2 Click inside the child class or interface and while continuing to hold down the mouse button, drag the cursor to the parent class or interface. Release the mouse button inside the parent class or interface.

The link appears between the two objects.

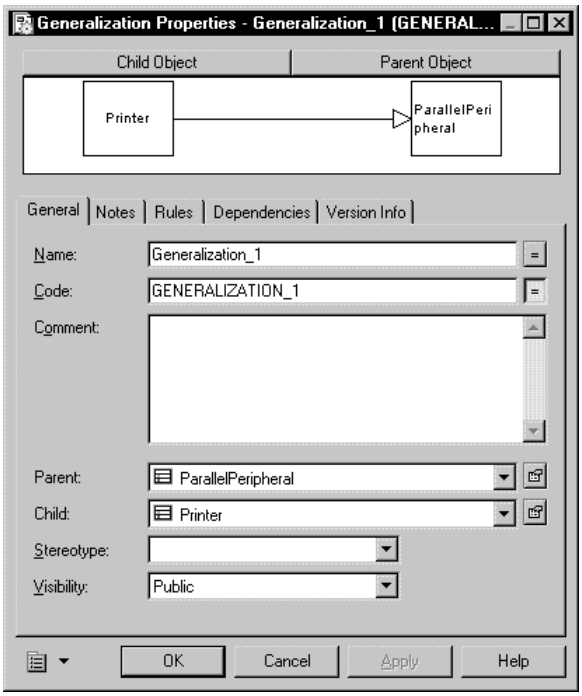
> **Dragging a generalization to a different class**
> You change the class or interface at either end of a generalization by clicking the generalization to select it, pressing down CTRL, and dragging one of the attach points to a different class or interface.

3 Click the pointer tool.
or
Click the right mouse button.

You release the Generalization tool.

4 Double-click the generalization in the diagram to display the generalization property sheet.

5 Type a name and a code.



You can open the property sheet of the child and parent objects by clicking the Child Object and Parent Object buttons located in the upper part of the generalization property sheet.

6 Click OK.

Modifying generalization properties in a class diagram

You can modify the generalization properties using the following methods:

- From the generalization property sheet.
- From the list of generalizations

For more information on the different ways to modify the generalization properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying generalization display preferences

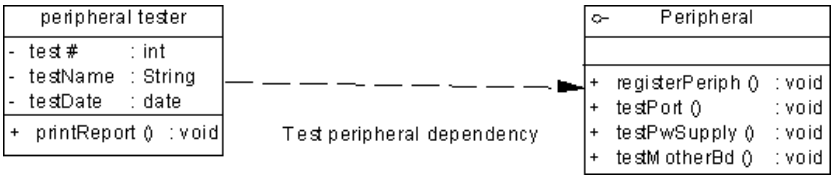
You can modify the following display preferences for generalizations using the Tools→Display Preferences command.

Preference	When selected
Show name	Displays the name of the generalization
Show stereotypes	Displays the stereotypes of the generalization
Show constraints	Displays the constraints (business rules) of the generalization

Defining dependencies in a class diagram

A **dependency** is a relationship between two modeling elements, in which a change to one modeling element (the influent element) will affect the other modeling element (the dependent element).

The dependency relationship indicates that one object in a diagram uses the services or facilities of another object. You can also define dependencies between a package and a modeling element.


peripheral tester
- test # : int
- testName : String
- testDate : date
+ printReport () : void
Test peripheral dependency
Peripheral
+ registerPeriph () : void
+ testPort () : void
+ testPwSupply () : void
+ testMotherBd () : void

Dependencies exist in the following diagrams:

- Class diagram

  A dependency can be created between:

  A class and an interface (and vice versa)

  Two classes

  Two interfaces

- Use case diagram

  For more information on dependencies in a use case diagram, see section Defining dependencies in a use case diagram, in chapter Building a Use Case Diagram.

- Component diagram

  For more information on dependencies in a component diagram, see section Defining dependencies in a component diagram, in chapter Building a Component diagram.

- Deployment diagram

  For more information on dependencies in a deployment diagram, see section Defining dependencies in a deployment diagram, in chapter Building a Deployment diagram.

- Object diagram

  For more information on dependencies in an object diagram, see section Defining dependencies in an object diagram, in chapter Building an Object Diagram.

Dependency properties in a class diagram

You can double-click any dependency symbol in a diagram to display its properties.

A dependency has the following properties:

Property	Description	Maximum length
Name	Name of the dependency	254
Code	Reference name of the dependency	254
Comment	Descriptive comment for the dependency	__
Influent	Class or interface influencing the dependent object. Changes on the influent object affect the dependent object	254
Dependent	Class or interface depending on the influent object. Changes on the dependent object will not affect the influent object	254
Stereotype	Extends the semantics of a dependency derived from existing dependencies but specific to your needs	254

Stereotype

A dependency has the following default stereotypes:

Stereotype	Description
Access	Public contents of the target package that can by accessed by the source package
Bind	Source object that instantiates the target template using the given actual parameters
Call	Source operation that invokes the target operation
Derive	Source object that can be computed from the target
Extend	Target object extends the behavior of the source object at the given extension point
Friend	Source object that has special visibility towards the target
Import	Everything declared public in the target object becomes visible to the source object, as if it were part of the source object definition
Include	Inclusion of the behavior of the first object into the behavior of the client object, under the control of the client object

Stereotype	Description
Instantiate	Operations on the source class create instances of the target class
Refine	The target object has a greater level of detail than the source object
Trace	Historical link between the source object and the target object
Use	Semantics of the source object are dependent on the semantics of the public part of the target object

You can select the following default values as stereotype for a dependency in Java:

Stereotype	Description
<<ejb-ref>>	Used in Java Generation to create references to EJBs (entity beans and session beans) for generating the deployment descriptor
<<sameFile>>	Used in Java Generation to generate Java classes of visibility protected or private within a file corresponding to a class of visibility public

Creating a dependency in a class diagram

You create a dependency between two classes, two interfaces, or between a class and an interface:

- From a diagram
- From the Browser

At creation, a dependency has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a dependency, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a dependency from a class diagram:**

1 Click the Dependency tool.

2 Click inside the dependent object and while continuing to hold down the mouse button, drag the cursor to the influent object. Release the mouse button inside the influent object.

The link appears between the two objects.

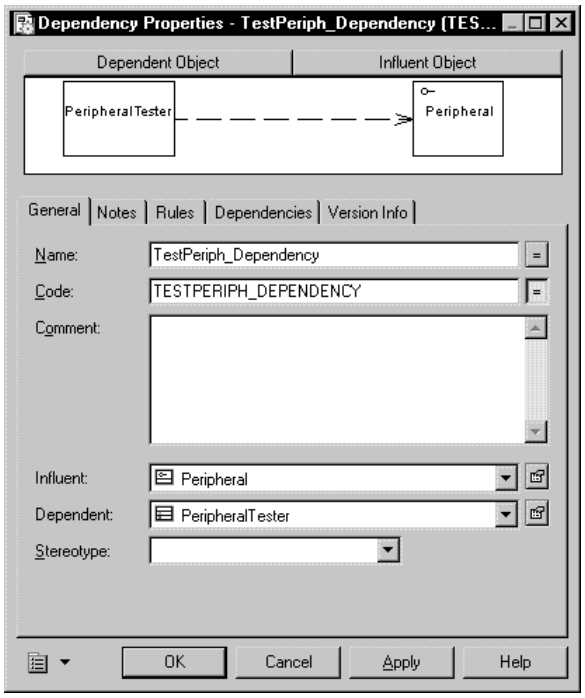
> **Dragging a dependency to a different object**
> You change the object at either end of a dependency by clicking the dependency to select it, pressing down CTRL, and dragging one of the attach points to a different object.

3 Click the Pointer tool.
or
Click the right mouse button to release the Dependency tool.

4 Double-click the dependency in the diagram to display the dependency property sheet.

For opening property sheets at last accessed page, see the framed text in Attaching an inner classifier to a class or an interface.

5 Type a name and a code.



You can open the property sheet of the dependent and influent objects by clicking the Dependent Object and Influent Object buttons located in the upper part of the dependency property sheet.

6 Click OK.

Modifying dependency properties in a class diagram

You can modify the dependency properties using the following methods:

- From the dependency property sheet
- From the list of dependencies

For more information on the different ways to modify the dependency properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying dependency display preferences

You can modify the following display preferences for dependencies using the Tools→Display Preferences command.

Preference	When selected
Show name	Displays the name of the dependency
Show stereotype	Displays the stereotype of the dependency
Show constraints	Displays the constraints (business rules) of the dependency

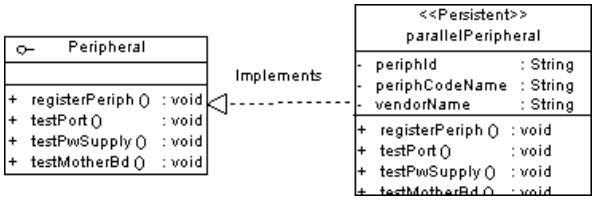
Defining realizations

A realization is a relationship between a class and an interface. In a realization, the class implements the methods specified in the interface. The interface is called the specification element and the class is called the implementation element.

You can also create a realization between a shortcut of an interface and a class, or between a shortcut of a class and a interface. Whenever the link is oriented, only the parent object can be the shortcut.

Although you can create more than one realization link between a class and an interface, you should create only one because the interface can only realize one action for the class. When you generate from an OOM, if more than one realization exists between a class and an interface, a warning message is generated.

The arrowhead at one end of the realization always points towards the interface.



Realization properties

A realization has the following properties:

Property	Description	Maximum length
Name	Name of the realization	254
Code	Reference name of the realization	254
Comment	Descriptive comment for the realization	—
Interface	Name of the interface that carries out the realization	254
Class	Name of the class for which the realization is carried out	254
Stereotype	Extends the semantics of a realization derived from existing realizations but specific to your needs	254

Creating a realization

You create a realization from a class to an interface:

- From a diagram
- From the Browser

At creation, a realization has a default name including a number, this number is assigned in the order of creation.

❖ **To create a realization from a diagram:**

1 Click the Realization tool.

2 Click inside the class and while continuing to hold down the mouse button, drag the cursor to the interface. Release the mouse button inside the interface.

The realization appears between the two objects.

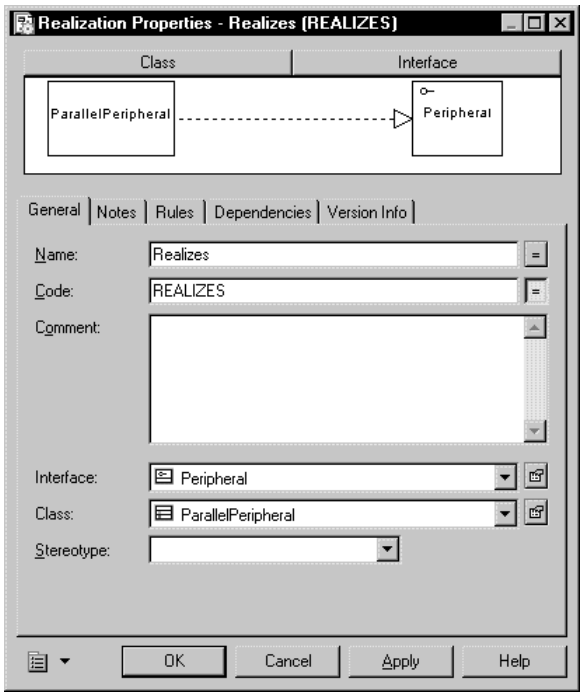
3 Click the Pointer tool.
or
Click the right mouse button.

You release the Realization tool.

4 Double-click the realization in the diagram to display the realization property sheet.

For opening property sheets at last accessed page, see the framed text in Attaching an inner classifier to a class or an interface.

5 Type a name and a code.



You can open the property sheet of the class and interface linked by the realization by clicking the Class and Interface buttons located in the upper part of the realization property sheet.

6 Type or select realization properties as required in the tabbed pages.

7 Click OK.

Modifying realization properties

You can modify the realization properties using the following methods:

- From the realization property sheet
- From the list of realizations

For more information on the different ways to modify the realization properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying realization display preferences

You can modify the following display preferences for realizations preferences using the Tools→Display Preferences command:

Preference	When selected
Show name	Displays the name of the realization
Show stereotypes	Displays the stereotypes of the realization
Show constraints	Displays the constraints (business rules) of the realization

Defining domains in an OOM

Domains define the set of values for which an attribute is valid. They are used to enforce consistent handling of data across the system. Applying domains to attributes makes it easier to standardize data characteristics for attributes in different classes.

In an OOM, you can associate the following information with a domain:

- Data type
- Check parameters
- Business rules

Domain properties in an OOM

Each domain includes the following properties:

Property	Description	Maximum length
Name	Name for the domain	254
Code	Reference name of the domain	254
Comment	Descriptive label for the domain	—
Stereotype	Allows the classification of the domain	254
Data type	Form of the data corresponding to the domain ; numeric, alphanumeric, Boolean, or others	—
Multiplicity	Specification of the range of allowable number of values attributes using this domain may hold	—
Persistent	Groupbox for valid generation of CDM or PDM persistent models. Defines a model as persistent	—

A domain also includes the following properties, which have associated values or information used by attributes attached to the class:

Property	Description
Standard checks	Check parameters defined for the domain
Additional checks	Domain constraints or validation rules not defined by standard check parameters
Rules	Business rules attached to the domain

Persistent domain in an OOM

The Detail page contains a Persistent groupbox whose purpose is to improve the generation of codes and data types during generation of a conceptual data model or a physical data model from an object-oriented model.

The Persistent groupbox contains the following values:

- Data type: specifies a persistent data type used in the generation of a persistent model, either a CDM or a PDM. The persistent data type is defined from default PowerDesigner conceptual data types
- Length: maximum number of characters of the persistent data type
- Precision: number of places after the decimal point, for persistent data type values that can take a decimal point

For more information on object persistence, see section Managing object persistence during OOM to PDM generation, in chapter Generating from an Object-Oriented Model.

Domain multiplicity in an OOM

The multiplicity of a domain is useful when working with a multiple attribute for example. The multiplicity is part of the data type and both multiplicity and data type may come from the domain. You can set the multiplicity as follows:

Multiplicity	Number of instances
0..1	Zero or one
0..*	None to unlimited
1..1	Exactly one
1..*	One to unlimited
*	None to unlimited

Creating a domain in an OOM

There are several ways to create a domain:

- From the Browser
- From the list of domains

At creation, a domain has a default name including a number, this number is assigned in the order of creation.

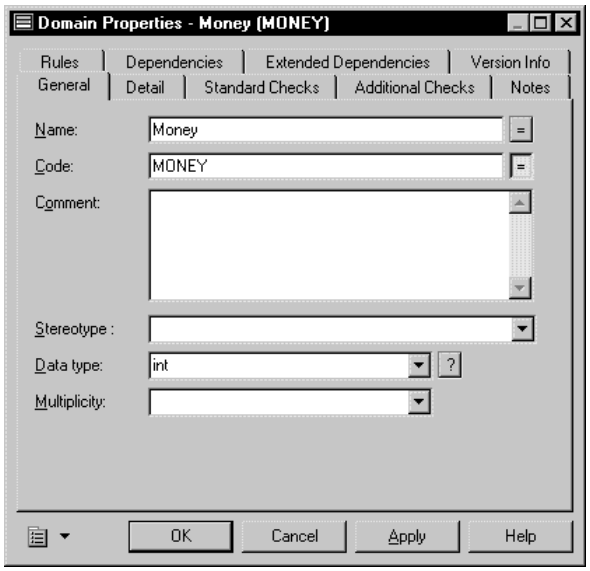
For more information on the different ways to create a domain, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a domain from the Browser:**

1 Right-click the model node in the Browser.

2 Select New→Domain to display the domain property sheet.

3 Type a name and a code.



4 Click OK.

The node of the new domain is visible in the Domains category of the Browser.

Selecting a data type for a domain in an OOM

The data types available in an OOM depend on your current object language.

You can select a data type for a domain in two ways:

- From the domain property sheet

- From the List of Domains

Selecting a data type for a domain from the property sheet

❖ **To select a data type for a domain from the property sheet:**

1. Select Model→Domains to display the list of domains.
2. Select the domain to define.

   An arrow appears at the beginning of the line.
3. Click the Properties tool.
   or
   Double-click the arrow at the beginning of the line.

   The domain property sheet appears.
4. Select a data type from the Data type dropdown listbox.
   or
   Click the Select a classifier button to select a data type that is a classifier.
5. Click OK.

   The change of data type appears in the list of domains.

Selecting a data type for a domain from the list of domains

❖ **To select a data type for a domain from the list of domains:**

1. Select Model→Domains to display the list of domains.
2. Select the domain you want to define.

   An arrow appears at the beginning of the line.
3. Click in the Data type column.

   A dropdown listbox appears.
4. Select a data type from the dropdown listbox.
5. Click OK.

Updating attributes using a domain in an OOM

When you modify a domain, you can choose to automatically update the following properties for attributes using the domain:

- Data type
- Check parameters
- Business rules

❖ **To update domain properties:**

1 Select Model→Domains to display the list of domains.

2 Click a domain from the list.

An arrow appears at the beginning of the line.

3 Click the Properties tool.
or
Double-click the arrow at the start of the line.

The domain property sheet appears.

> **Accessing a property sheet from the Browser**
> You also access a domain property sheet by double-clicking the appropriate domain node in the Browser.

4 Type or select domain properties as required in the tabbed pages.

5 Click OK.

If the domain is used by one or more attributes, an update confirmation box asks you if you want to update Data type and Check parameters for the attributes using the domain.

6 Select the properties you want to update for all attributes using the domain.

7 Click Yes.

Defining check parameters

Check parameters are sets of conditions which data must satisfy to remain valid. They are attached to attributes and domains. They are principally used in a CDM or a PDM. They are of importance only when importing a CDM or importing a PDM into an OOM. They are not used in the OOM alone.

There are three types of check parameters you can therefore use in an OOM:

Parameter type	Description
Standard parameters	Common data constraints defining a data range
Additional check parameters	SQL expression defining a data constraint using variables instantiated with standard parameter values
Validation rule	Business rule defined as a server expression

Setting standard check parameters for OOM objects

Standard parameters indicate common data constraints.

Parameter	Description
Minimum	Lowest acceptable numeric value
Maximum	Highest acceptable numeric value
Default	Value assigned in absence of an expressly entered value
Format	Data format (for example, 9999.99)
Unit	Standard measure
Uppercase	Forces all alphabetical characters to uppercase
Lowercase	Forces all alphabetical characters to lowercase
Cannot Modify	Protects from changes, results in a non modifiable attribute
List of Values	Authorized values

❖ **To set standard parameters:**

1 Click the Standard Checks tab in the property sheet of a domain or an attribute to display the Standard Checks page.

2 Select your Standard parameters as required.

3 Click OK.

Defining additional check parameters for OOM objects

You can use the following variables as check parameters:

Variable	Description
%MINMAX%	Minimum and maximum values defined in Values groupbox on Standard Checks page
%LISTVAL%	Customized values defined in List Values groupbox on Standard Checks page
%RULES%	Validation rule expression defined on Expression page of the Rules property sheet

You define additional check parameters for data constraints where standard check parameters are not sufficient.

❖ **To define additional check parameters:**

1 Select Model→Domains.

The List of Domains appears.

2 Click a domain in the list.

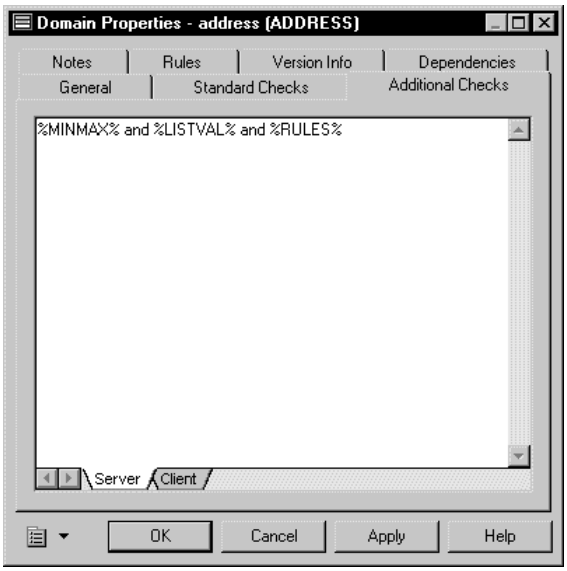
An arrow appears at the beginning of the line.

3 Click the Properties tool.

The property sheet for the domain appears.

4 Click the Additional Checks tab.

The Additional Checks page appears.



5 Type expressions using the variables %MINMAX%, %LISTVAL%, and %RULES%.

6 Click OK.

CHAPTER 4

Building an Object Diagram

About this chapter

This chapter describes how to build an object diagram. It explains the role of each element in an object diagram and how to create and modify them.

Contents

Topic	Page
Object diagram basics	174
Defining objects in an object diagram	176
Defining attribute values	181
Defining instance links in an object diagram	185
Defining dependencies in an object diagram	190

Object diagram basics

The **object diagram** and the class diagram constitute the two UML static structure diagrams. The object diagram is considered a diagram of instances of classes.

Defining an object diagram

The object diagram is one of the diagrams of the Unified Modeling Language (UML). The object diagram describes the structure of model elements and not their behavior or temporary information. It is a diagram of instances: it represents instances of class (objects), instances of association (instance links), and dependencies. However, it does not represent inheritance relationships.

As a diagram of instances, the object diagram shows an example of data structures with data values that corresponds to a detailed situation of the system at a particular point in time.

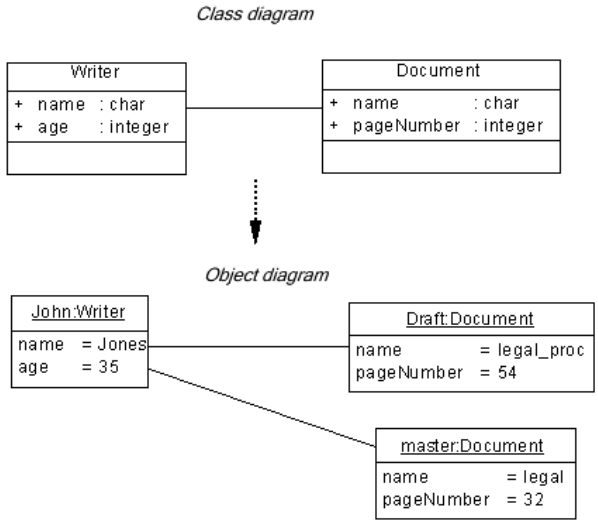
Why build an object diagram?

The object diagram can be used for analysis purpose: constraints between classes that are not classically represented in a class diagram can typically be represented in an object diagram.

If you are novice in object modeling, instances usually have more meaning than classifiers do, because classifiers represent a level of abstraction. Gathering several instances under the same classifier helps you to understand what classifiers are. Moreover, even for analysts used to abstraction, the object diagram can help understand some structural constraints that cannot be easily graphically specified in a class diagram.

In this respect, the object diagram is a limited use of a class diagram. For example, see the following figures representing a class and an object diagram: the class diagram specifies that a class Writer is linked to a class Document.

The object diagram, deduced from this class diagram, highlights some of the following details: the object named John, instance of the class Writer is linked to two different objects Draft and Master that are both instances of the class Document.



Dragging classes and associations to an object diagram
You can drag classes and associations from the Browser and drop them into an object diagram. If you drag classes, new objects as instances of classes are created. If you drag an association, a new instance link as instance of the association, and two objects are created.

For more information on diagrams of the OOM, see section What is an OOM, in chapter Object-Oriented Model Basics.

Creating an object diagram

You can create an object diagram using different methods.

For more information on creating an object diagram in a new OOM, see section Creating an OOM, in chapter Object-Oriented Model Basics.

For more information on creating an object diagram in an existing OOM, see section Creating a new diagram, in chapter Managing Models, in the *General Features Guide*.

Defining objects in an object diagram

At the conceptual level, an **object** is an element defined as being part of the system described. It represents an object that has not yet been instantiated because the classes are not yet clearly defined at this stage.

If you need to go further with the implementation of your model, the object that has emerged during analysis will probably turn into an instance of a defined class. In this case, an object is considered an instance of a class.

Three possible situations can be represented:

When an object is not an instance of a class it has only a name.

When an object is an instance of a class it has a name and a class.

When an object is an instance of a class but is actually representing any or all instances of a class it has a class but no name.

In the object diagram, an object instance of a class can display the values of attributes defined on the class. When the class is deleted, the associated objects are not deleted.

For more information on attribute values, see section Defining attribute values.

In the object diagram, an object can be Multiple when it defines a set of instances. It is a graphical representation of an object that represents several instances. An object can communicate with another object that is a multiple. This feature is mainly used in the collaboration diagram but can also be used in the object diagram.

When the Multiple check box is selected in the object property sheet, a specific symbol with two superposed rectangles appears.



The object shares the same concept in the object, sequence and collaboration diagrams. It can either be created in the diagram type you need, or dragged from a diagram type and dropped into another diagram type.

For more information on objects in the sequence diagram, see section Defining objects in a sequence diagram, in chapter Building a sequence diagram.

For more information on objects in the collaboration diagram, see section Defining objects in a collaboration diagram, in chapter Building a collaboration diagram.

Object properties in an object diagram

You can double-click any object symbol in a diagram to display its properties.

An object has general properties:

Property	Description	Maximum length
Name	Name of the object. Not compulsory	254
Code	Reference name or code for the object. Not compulsory	254
Comment	Descriptive comment for the object	—
Stereotype	Extends the semantics of an object derived from existing objects but specific to your needs	254
Class	Class from which an object is an instance	254
Multiple	Object that represents several instances	—

Name and code

The name and the code are not compulsory since you can have an object representing an unnamed instance of a class, but if the name or code boxes are empty, you need to supply the class name in the Class box.

The name and the code should be unique per class. Two objects could share the same name only if they belong to different classes.

Class property of the object

You can drag a class from the Browser or from a class diagram and drop it into the object diagram. You can also copy a class and paste it, or paste it as shortcut, into the object diagram. This creates an object as mere instance of the class.

For more information on turning a class into an object, see section Linking a class to an object, in chapter Working with Object-Oriented Models.

Object attribute values

When an object is an instance of a class, the attributes of the class can be added to the object from the Attribute Values page in the object property sheet. Once the attribute is added, you can attach a value to the attribute from this same page.

When you add a value to the attribute, the attribute property sheet displays this information in the Attribute Values tabbed page of the Dependencies page.

For more information on attribute values, see section Defining attribute values.

Creating an object in an object diagram

There are several ways to create an object:

- From a diagram
- From the List of Objects

At creation, an object has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create an object, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ To create an object from an object diagram:

1 Click the Object tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



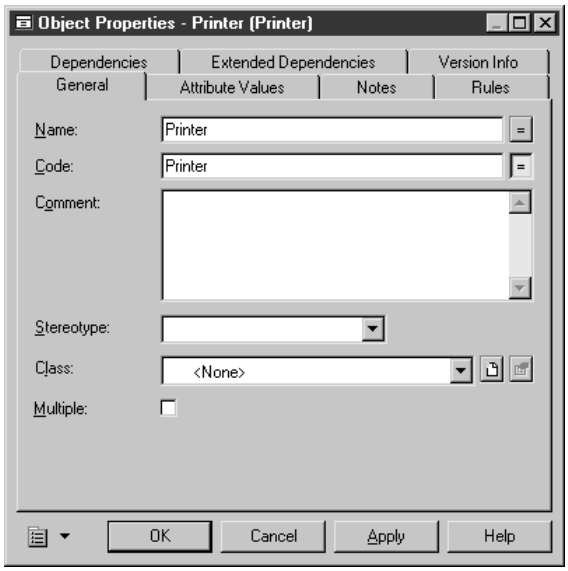
3 Click the Pointer tool.
or
Click the right mouse button.

You release the Object tool.

4 Double-click the object symbol to display its property sheet.

Opening property sheets at last accessed page
Property sheets open to the General page by default. However, you can choose to open property sheets at the last page accessed by selecting Tools→General Options→Dialog, and selecting the option Keep Last Tab in the Property Sheets groupbox.

5 Type a name and a code.



6 Click OK.

The object node appears in the Objects category of the Browser.

Modifying object properties in an object diagram

You can modify the object properties using the following methods:

- From the object property sheet
- From the List of Objects

For more information on the different ways to modify the object properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

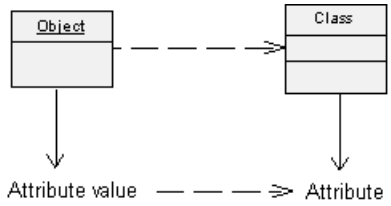
Modifying object display preferences

You can modify the following display preferences for objects using the Tools→Display Preferences command:

Preference	Description
Show stereotype	Displays the stereotype of the object
Show attribute values	Displays attributes and attribute values of the object

Defining attribute values

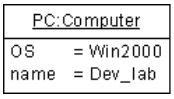
An attribute value represents an instance of a class attribute. It refers to an attribute of the class from which the current object is an instance, or it can refer to an attribute inherited from a parent of the class. An attribute value is stored as text.



An attribute value is created when you assign a value to an attribute. There is no symbol of attribute value, there is no property sheet either.

You access attribute values from the Attribute Values page in the object property sheet. An Add Attribute Values tool allows you to add attributes to this page. The Add Attribute Values tool opens a selection dialog box that displays all attributes of the class of the object, including inherited attributes of classes from which the class inherits (also known as **super classes)**. Once the attribute is added to the object, it is possible to define its value from the Value column in the list.

There is only **one** attribute value per attribute (attribute of the class of the object). Attribute values are displayed within the object symbol in the object diagram, if they are not displayed, you should modify the object display preferences.



You can assign a value to a migrated attribute.

For more information on migrated attribute, see section Migrating association roles in a class diagram, in chapter Building a class diagram.

You can not create shortcuts of attribute values, neither move nor copy attribute values. For all external shortcuts of objects, the list of attribute values is displayed only when the model containing the shortcut of the object is loaded in the workspace.

Attribute values behavior

When an attribute value is assigned to an object, it is associated with this object and the attribute must belong to the class of the object or to one of its inherited class.

The coherence between the class of the object, the attribute, and the attribute value is handled in PowerDesigner. For example when you change the inheritance, the attribute value on the corresponding object is automatically removed if the attribute value is not associated with attributes owned by the new class, or by its super classes.

The coherence between the order of attributes in the class and the order of attribute values is not handled. This allows you to sort the attribute values according to your needs.

The following rules apply to attribute values:

- When the class from which the object is an instance is an external shortcut, the attribute values of this object must be updated if the attributes of the class, or of the super classes have been modified, deleted, added or their data type changed. You have to update them after the model containing the class, or the model containing the object is loaded in the workspace
- When you remove an attribute, its corresponding attribute value is removed

Attribute value properties

An attribute value has no property sheet of its own. All properties displayed in the Attribute Values page of the object property sheet are calculated, read-only properties that are duplicated from the attribute. The only property that is not read-only is the Value property that you must type or select in order to create the attribute value.

You can display all properties in the Attribute Values page by selecting the Customize Columns and Filter tool:



Property	Description	Maximum length
Class	Class of the object. You can select the Ellipsis button in the Class column to display the class property sheet	254
Attribute	Attribute of the class of the object. You can select the Ellipsis button in the Attribute column to display the attribute property sheet	254

Property	Description	Maximum length
Value	Value of the attribute. You can type your own value, or select the down arrow and choose a value from the list. This list comes from the List of Values groupbox in the Standard Checks page of the attribute property sheet. If you use multi-valued attributes, you can use the syntax of your choice to type the values in this column	254
Data type	Data type of the attribute	—
Initial value	Initial value of the attribute (including quotes if needed)	—
Primary identifier	If selected, indicates that the attribute is part of a primary identifier For more information on primary identifiers, see section Defining identifiers, in chapter Building a class diagram	—

Creating an attribute value

You must create an attribute from the object property sheet in order to create a value for the attribute.

When you add an attribute, the Value column in the Attribute Values page appears empty.

❖ **To create an attribute value:**

1 Double-click the object symbol to display its property sheet.

2 Click the Attribute Values tab to display the Attribute Values page.

3 Click the Add Attribute Values tool to display a selection window.

4 Select the attribute you want to attach to the object.

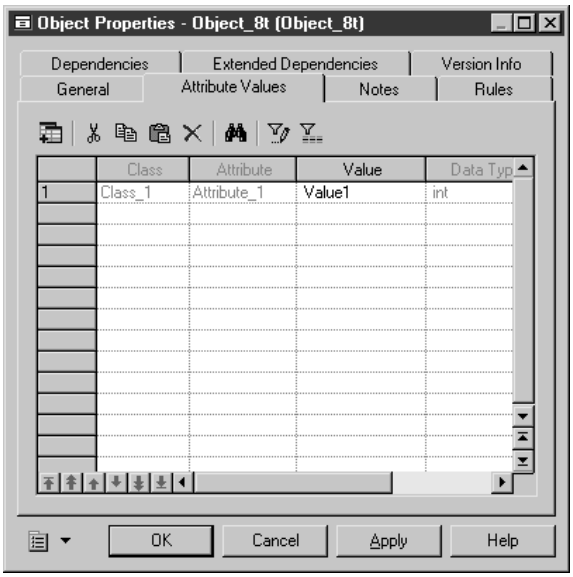
5 Click OK.

6 Select a value from the list by clicking the down arrow in the Value column.
or
Type a value directly in the Value column.

7 Click OK to close the object property sheet.

Modifying attribute value properties

You can modify the attribute value properties from the Attribute Values page of the object property sheet. If you want to display all properties of the attribute value, select the Customize Columns and Filter tool in the Attribute Values page, and select the items that you want to display in the list.



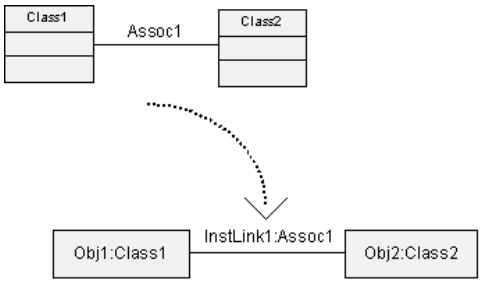
Defining instance links in an object diagram

An **instance link** represents a connection between two objects. It is drawn as a solid line between two objects.

The instance link in the object diagram shares the same concept as in the collaboration diagram.

For more information on instance links in the collaboration diagram, see section Defining instance links in a collaboration diagram, in chapter Building a collaboration diagram.

Instance links have a strong relationship with associations of the class diagram: associations between classes can become instance links (instances of associations) between objects (instances of classes) in the object diagram. Moreover, the instance link symbol in the object diagram is similar to the association symbol in the class diagram, except that the instance link symbol has no cardinalities.

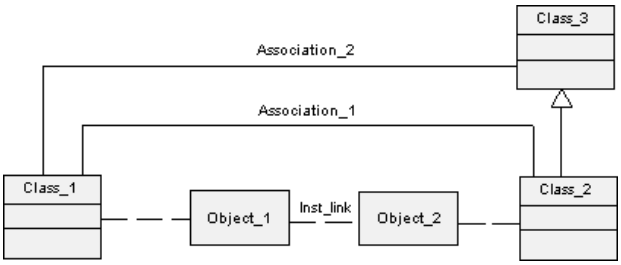


The roles of the instance link are duplicated from the roles of the association. An instance link can therefore be an aggregation or a composition, exactly like an association of the class diagram. If it is the case, the composition or aggregation symbol is displayed on the instance link symbol. The roles are displayed on the instance link symbol provided you select the Show Role Names display preference in the Instance Link category.

Example

The following figure shows Object_1 as instance of Class_1, and Object_2 as instance of Class_2. They are linked by an instance link. It shows Class_1 and Class_2 linked by an association. Moreover, since Class_2 is associated with Class_1 and also inherits from Class_3, there is an association between Class_1 and Class_3.

The instance link between Object_1 and Object_2 in the figure can represent Association_1 or Association_2.



You can also use shortcuts of associations, however you can only use it if the model to which the shortcut refers is open in the workspace.

Instance links behavior

The following rules apply to instance links:

- When an association between classes becomes an instance link, both classes linked by the association, and both classes of the objects linked by the instance link must match (or the class of the object must inherit from the parent classes linked by the association)
- Two instance links can be defined between the same source and destination objects (**parallel instance links**). If you merge two models, the Merge Model feature differentiates parallel instance links according to their class diagram associations
- You can use reflexive instance links (same source and destination object)

Instance link properties in an object diagram

You can double-click any instance link symbol in a diagram to display its properties:

An instance link has the following general properties:

Property	Description	Maximum length
Name	Name of the instance link. Calculated, read-only value	254
Code	Reference code of the instance link. Calculated, read-only value	254
Comment	Descriptive comment for the instance link	—

Property	Description	Maximum length
Stereotype	Extends the semantics of an instance link derived from existing instance links but specific to your needs	254
Object A	Name of the object at one end of the instance link. You can click the Properties tool beside this box to display the object property sheet	254
Object B	Name of the object at the other end of the instance link. You can click the Properties tool beside this box to display the object property sheet	254
Association	Association between classes that the instance link uses to communicate between objects of these classes. You can click the Properties tool beside this box to display the association property sheet	254

Drag and drop associations in an object diagram

You can drag an association from the Browser and drop it into an object diagram. This creates an instance link and two objects, instances of these classes.

Creating an instance link in an object diagram

You create an instance link between two objects:

- From a diagram
- From the Browser

Instance links name and code are calculated. The name and code boxes display a read-only value named after both extremities of the instance link.

For more information on the different ways to create an instance link, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an instance link from an object diagram:**

1 Click the Instance link tool.

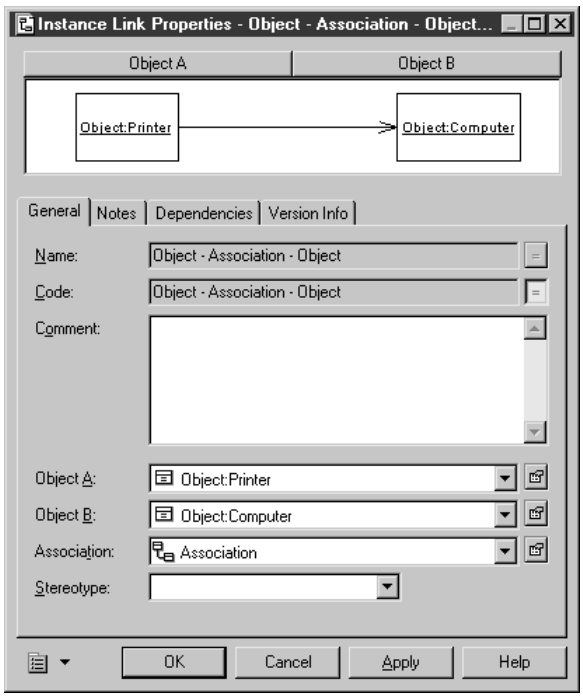
2 Click inside the first object and while continuing to hold down the mouse button, drag the cursor to the second object. Release the mouse button inside the second object.

The instance link appears between both objects.

3 Click the pointer tool.
or
Click the right mouse button.

You release the Instance link tool.

4 Double-click the instance link in the diagram to display the instance link property sheet.



You can open the property sheet of both objects connected by the instance link by clicking the Object A and Object B buttons located in the upper part of the instance link property sheet.

5 Click OK.

Modifying instance link properties in an object diagram

You can modify the instance link properties using the following methods:

- From the instance link property sheet
- From the list of instance links

For more information on the different ways to modify the instance link properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying instance link display preferences

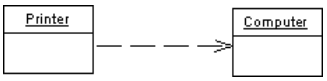
You can modify the following display preferences for instance links using the Tools→Display Preferences command.

Preference	When selected
Show association name	Displays the name of the association that the instance link refers to
Show stereotype	Displays the stereotype of the instance link
Show association role names	Displays the names of the association roles

Defining dependencies in an object diagram

A **dependency** is a relationship between two modeling elements, in which a change to one modeling element (the influent element) will affect the other modeling element (the dependent element).

The dependency relationship indicates that one object in a diagram uses the services or facilities of another object. You can also define dependencies between a package and a modeling element.



Dependency properties in an object diagram

You can double-click any dependency symbol in a diagram to display its properties:

Property	Description	Maximum length
Name	Name of the dependency	254
Code	Reference name of the dependency	254
Comment	Descriptive comment for the dependency	__
Influent	Object influencing the dependent object. Changes on the influent object affect the dependent object	254
Dependent	Object depending on the influent object. Changes on the dependent object will not affect the influent object	254
Stereotype	Extends the semantics of a dependency derived from existing dependencies but specific to your needs	254

Creating a dependency in an object diagram

You create a dependency between two objects:

- From a diagram
- From the Browser

At creation, a dependency has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a dependency, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a dependency from an object diagram:**

1 Click the Dependency tool.

2 Click inside the dependent object and while continuing to hold down the mouse button, drag the cursor to the influent object. Release the mouse button inside the influent object.

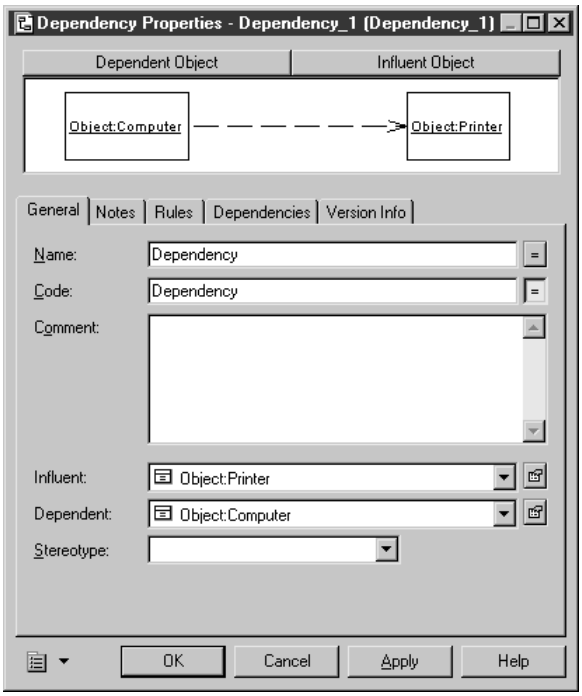
The dependency appears between both objects.

> **Dragging a dependency to a different object**
> You change the object at either end of a dependency by clicking the dependency to select it, pressing down CTRL, and dragging one of the attach points to a different object.

3 Click the Pointer tool.
or
Click the right mouse button to release the Dependency tool.

4 Double-click the dependency in the diagram to display the dependency property sheet.

5 Type a name and a code.



6 Click OK.

Modifying dependency properties in an object diagram

You can modify the dependency properties using the following methods:

- From the dependency property sheet
- From the list of dependencies

For more information on the different ways to modify the dependency properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying dependency display preferences

You can modify the following display preferences for dependencies using the Tools→Display Preferences command.

Preference	When selected
Show name	Displays the name of the dependency
Show stereotype	Displays the stereotype of the dependency
Show constraints	Displays the constraints (business rules) of the dependency

CHAPTER 5

Building a Collaboration Diagram

About this chapter

This chapter describes how to build a collaboration diagram. It explains the role of each element in a collaboration diagram and how to create and modify them.

Contents

Topic	Page
Collaboration diagram basics	196
Defining actors in a collaboration diagram	199
Defining objects in a collaboration diagram	203
Defining instance links in a collaboration diagram	207
Defining messages in a collaboration diagram	213

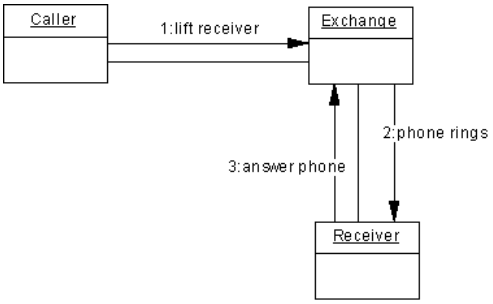
Collaboration diagram basics

The **collaboration diagram** is part of the Unified Modeling Language (UML). It is one of the two interaction diagrams, the other being the sequence diagram.

Defining a collaboration diagram

The collaboration diagram is part of an Object-Oriented Model. It is one of the available diagrams in an OOM. It shows actors, objects (instances of classes) and their communication links (called instance links), as well as messages sent between them.

A collaboration diagram shows these elements performing a particular goal (a functionality of the system). It may be used to illustrate an execution of an operation, a use-case execution, or simply an interaction scenario in the system. It designs an example of interactions between objects.



Collaboration diagrams convey the same information as sequence diagrams. The main difference is that the collaboration diagram focuses on objects in action, it shows a network of objects that are collaborating (focusing on object structure) whereas the sequence diagram focuses on interactions based on chronology.

For more information on the sequence diagram, see chapter Building a sequence diagram.

Why build a collaboration diagram?

A collaboration diagram represents behavior in terms of interactions. It complements the class diagram that represents the static structure of the system by specifying the behavior of classes, interfaces, and the possible use of their operations.

Building a collaboration diagram can also be the opportunity to test the static model at the conception level; it may represent a scenario in which classes from the class diagram are instantiated to create the objects necessary to run the scenario.

A collaboration diagram can also be used to model a use case behavior, refine a use case description, or refine a class diagram using an iterative approach. This approach is useful during requirement analysis because it may help find additional classes and additional objects that may not have emerged at the very beginning.

Applicability

Collaboration diagrams are particularly important for designers because they clarify the roles of objects. You may build a collaboration diagram before a sequence diagram (if you plan to build both interaction diagrams), and after completion of a class diagram to illustrate the interactions of objects derived from the classes. You may use one or more collaboration diagrams to enact a use case, or to split up a complex behavior into logical subdivisions.

For more information on diagrams of the OOM, see section What is an OOM, in chapter Object-Oriented Model Basics.

Creating a collaboration diagram

You can create a collaboration diagram using different methods.

For more information on creating a collaboration diagram in a new OOM, see section Creating an OOM, in chapter Object-Oriented Model Basics.

For more information on creating a collaboration diagram in an existing OOM, see section Creating a new diagram, in chapter Managing Models, in the *General Features Guide*.

Converting a collaboration diagram to a sequence diagram

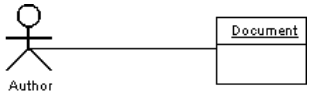
You can convert a collaboration diagram to a sequence diagram.

To do so, create a sequence diagram at the same level as the existing collaboration diagram (under the same package), right-click the diagram background and select Diagram→Show Symbols from the contextual menu. This opens a selection dialog box from which you can select objects of the collaboration diagram that will be used into the sequence diagram.

You can also use a Visual Basic script that converts the existing collaboration diagram to a sequence diagram. To do so, run the script named CollaborationToSequence.VBS from the collaboration diagram by selecting Tools→Execute Commands→Edit/Run scripts. The script is located in the VB Scripts folder of the PowerDesigner installation.

Defining actors in a collaboration diagram

The system you are describing interacts with **actors**. The definition of an actor is the same as in a use case, or a sequence diagram, it characterizes an outside user, or a related set of users that interact with a system.



Actors can be shared between use case, sequence and collaboration diagrams if they play the same role. Actors are common to the model. They can either be created in the diagram type you need, or dragged from a diagram type and dropped into another diagram type.

For more information on actors in the use case diagram, see section Defining actors in a use case diagram, in chapter Building a use case diagram.

For more information on actors in the sequence diagram, see section Defining actors in a sequence diagram, in chapter Building a sequence diagram.

Actor properties in a collaboration diagram

You can double-click any actor symbol in a diagram to display its properties.

An actor has the following general properties:

Property	Description	Maximum length
Name	Name of the actor	254
Code	Reference name or code for the actor	254
Comment	Descriptive comment for the actor	—
Stereotype	Extends the semantics of an actor derived from existing actors but specific to your needs	254

Actor implementation classes

The Implementation Classes page shows the list of classes and interfaces used to implement an actor.

For more information on implementation classes, see section Linking a class or interface to an actor, in chapter Working with Object-Oriented models.

Actor related diagrams

In the actor property sheet, you can select the Related Diagrams tab to display all diagrams of the model that are related to an actor. They can be attached and detached from the current actor at this stage. This page displays a list of the diagrams identified by their name and type.

For more information on Related Diagrams, see section Defining related diagrams, in chapter Working with Object-Oriented models.

Creating an actor in a collaboration diagram

There are several ways to create an actor:

- From a diagram
- From the Browser
- From the List of Actors

At creation, an actor has a default name including a number, this number is assigned in the order of creation.

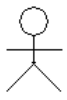
For more information on the different ways to create an actor, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an actor from a collaboration diagram:**

1 Click the Actor tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



3 Click the Pointer tool.
or
Click the right mouse button.

You release the Actor tool.

4 Double-click the actor symbol to display its property sheet.

5 Type a name and a code.



6 Click OK.

The actor node appears in the Actors category of the Browser.

Showing the symbol of an actor

You can show the symbol of an actor already created in a use case, or sequence diagram using different methods.

For more information on how to show an actor symbol, see section Showing the symbol of an actor, in chapter Building a use case diagram.

Modifying actor properties in a collaboration diagram

You can modify the actor properties using the following methods:

- From the actor property sheet
- From the List of Actors

For more information on the different ways to modify the actor properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying actor display preferences

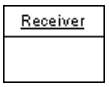
You can modify the following display preferences for actors using the Tools→Display Preferences command.

Preference	Description
Show stereotype	Displays the stereotype of the actor

Defining objects in a collaboration diagram

An **object** is an instance of a class. It can be persistent or transient: persistent is the situation of an object that continues to exist after the process that created it has finished, and transient is the situation of an object that stops to exist when the process that created it finishes.

The name of the object is displayed underlined. The Underline character traditionally indicates that an element is an instance of another element.



The object shares the same concept in the object, sequence and collaboration diagrams. It can either be created in the diagram type you need, or dragged from a diagram type and dropped into another diagram type.

For more information on objects in the object diagram, see section Defining objects in an object diagram, in chapter Building an object diagram.

For more information on objects in the sequence diagram, see section Defining objects in a sequence diagram, in chapter Building a sequence diagram.

Defining multiples

A **multiple** defines a set of instances. It is a graphical representation of an object that represents several instances, however it can only hold one set of attributes even if it represents several instances. An object can communicate with another object that is a multiple. This feature is mainly used in the collaboration diagram but can also be used in the object diagram.

Example

A clerk handles a list of documents: it is the list of documents that represents a multiple object.

When the Multiple check box is selected in the object property sheet, a specific symbol (two superposed rectangles) appears.



Object properties in a collaboration diagram

You can double-click any object symbol in a diagram to display its properties:

Property	Description	Maximum length
Name	Name of the object. Not compulsory	254
Code	Reference name or code for the object. Not compulsory	254
Comment	Descriptive comment for the object	—
Stereotype	Extends the semantics of an object derived from existing objects but specific to your needs	254
Class	Class of which an object is an instance You can link an object to a class, or create a new class from this dropdown listbox	254
Multiple	Object that represents several instances	—

Name and code

The name and the code are not compulsory since you can have an object representing an unnamed instance of a class, but if the name or code boxes are empty, you need to supply the class name in the Class box.

The name and the code should be unique per class. Two objects could share the same name only if they belong to different classes.

Creating an object in a collaboration diagram

There are several ways to create an object:

- From a diagram
- From the Browser
- From the List of Objects

At creation, an object has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create an object, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an object from a collaboration diagram:**

1 Click the Object tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



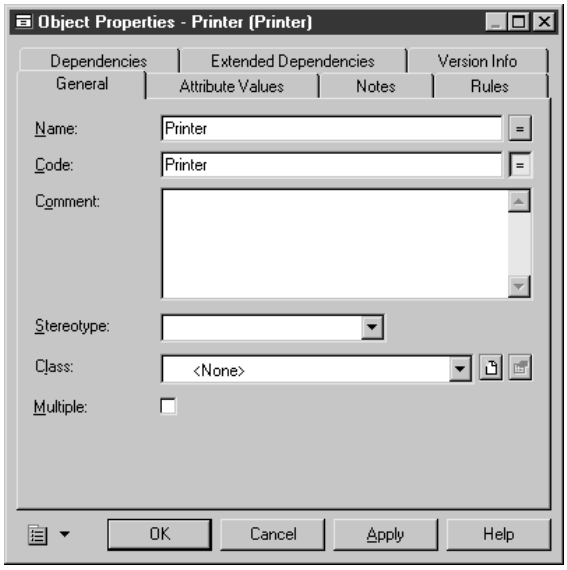
3 Click the Pointer tool.
or
Click the right mouse button.

You release the Object tool.

4 Double-click the object symbol to display its property sheet.

> **Opening property sheets at last accessed page**
> Property sheets open to the General page by default. However, you can choose to open property sheets at the last page accessed by selecting Tools→General Options→Dialog, and selecting the option Keep Last Tab in the Property Sheets groupbox.

5 Type a name and a code.



6 Click OK.

The object node appears in the Objects category of the Browser.

Turning a class into an object
You can drag a class node from the Browser, or drag a class from a class diagram and drop it into the collaboration diagram. You can also copy a class and paste it, or paste it as shortcut, into the collaboration diagram. This automatically creates an object, instance of the class.

Modifying object properties in a collaboration diagram

You can modify the object properties using the following methods:

- From the object property sheet
- From the List of Objects

For more information on the different ways to modify the object properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying object display preferences

You can modify the following display preferences for objects using the Tools→Display Preferences command.

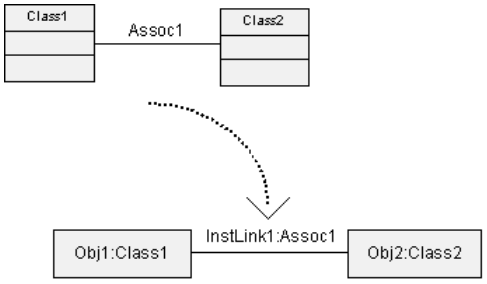
Preference	Description
Show stereotype	Displays the stereotype of the object
Show attribute values	Displays the attribute values of the object

Defining instance links in a collaboration diagram

An instance link represents a connection between objects, it highlights the collaboration between objects, hence the name ‘collaboration diagram’. It is drawn as a solid line between:

- Two objects
- An object and an actor (and vice versa)

An instance link can be an instance of an association between classes. This can help you link an object to another object:

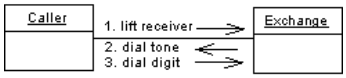


The instance link in the collaboration diagram shares the same concept as in the object diagram.

For more information on instance links in the object diagram, see section Defining instance links in an object diagram, in chapter Building an object diagram.

The role of the instance link comes from the association. The name of an instance link comprises the names of both objects at the extremities, plus the name of the association.

The symbol of the instance link may contain several message symbols attached to it.



Instance links hold an ordered list of messages: the **sequence number** that specifies the order in which messages are exchanged between objects.

For more information on sequence numbers, see sections Creating a sequence number in a collaboration diagram and Moving sequence numbers in a collaboration diagram.

Instance links behavior

The following rules apply to instance links:

- You can use a recursive instance link with an object (same source and destination object)
- Two instance links can be defined between the same source and destination objects (**parallel instance links**)
- When you delete an instance link, its messages are also deleted if no sequence diagram already uses these messages
- When an association between classes turns into an instance link, both classes linked by the association, and both classes of the objects linked by the instance link must match (or the class of the object must inherit from the parent classes linked by the association)
- If you change one end of an association, the instance link that comes from the association is detached
- When you copy and paste, or move an instance link, its messages are automatically copied at the same time
- When the extremities of the message change, the message is detached from the instance link
- If you use the Show Symbols feature to display an instance link symbol, all the messages attached to the instance link are displayed

Instance link properties in a collaboration diagram

You can double-click any instance link symbol in a diagram to display its properties:

Property	Description	Maximum length
Name	Name of the instance link. Calculated, read-only value	254
Code	Reference code of the instance link. Calculated, read-only value	254
Comment	Descriptive comment for the instance link	__
Object A	Name of the object at one end of the instance link. You can click the Properties tool beside this box to display the object property sheet	254
Object B	Name of the object at the other end of the instance link. You can click the Properties tool beside this box to display the object property sheet	254
Association	Association that the instance link uses between objects. You can click the Properties tool beside this box to display the association property sheet	254
Stereotype	Extends the semantics of an instance link derived from existing instance links but specific to your needs	254

Messages page of the instance link

You can add or edit a message from the Messages page in the property sheet of the instance link using the following tools:

Tool	Description
	Create a New Message tool. It allows you to create a message attached to the instance link. This message has a default sender and receiver, and a default sequence number. The message can be vertical or horizontal in the diagram depending on the position of the instance link to which it is attached
	Reuse a Message. Allows you to display all messages of the same extremities as the current instance link that are not used by another instance link (it may be created and used in a sequence diagram). This tool does not allow you to create a copy of the message but to create a new symbol of the message

Tool	Description
	Add Messages. Opens a selection window to select messages to be added. This selection window displays all the messages within the model. When you select this tool, you make a copy of the selected message, and Object A and Object B are set as default sender and receiver of the message

Accessing the Messages page
You can access the Messages page of an instance link by right clicking the instance link symbol in the diagram, and selecting Messages from the contextual menu.

Creating an instance link in a collaboration diagram

You create an instance link between two objects:

- From a diagram
- From the Browser

At creation, an instance link has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create an instance link, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an instance link from a collaboration diagram:**

1 Click the Instance Link tool.

2 Click inside the first object and while continuing to hold down the mouse button, drag the cursor to the second object. Release the mouse button inside the second object.

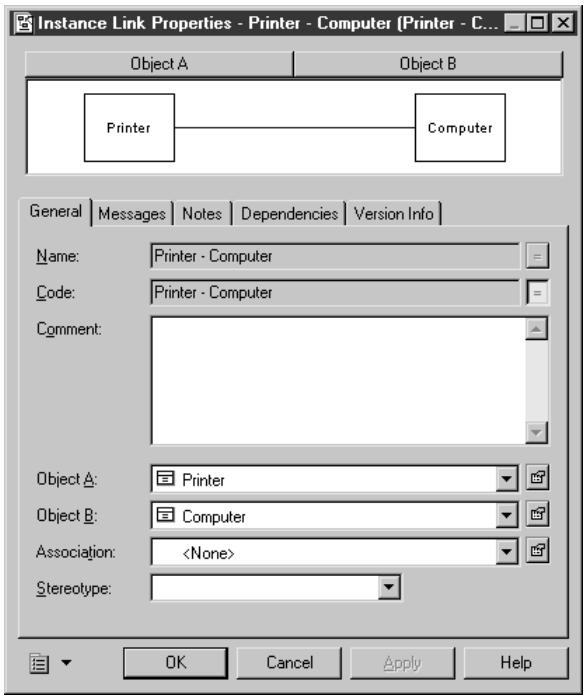
The instance link appears between both objects.

3 Click the pointer tool.
or
Click the right mouse button.

You release the Instance Link tool.

4 Double-click the instance link in the diagram to display the instance link property sheet.

5 Type a name and a code.



You can open the property sheet of both objects connected by the instance link by clicking the Object A and Object B buttons located in the upper part of the property sheet of the instance link.

6 Click OK.

Drag and drop associations
You can drag an association from the Browser and drop it into a collaboration diagram. This creates an instance link as instance of the association, and two objects as instances of these classes

Modifying instance link properties in a collaboration diagram

You can modify the instance link properties using the following methods:

- From the instance link property sheet
- From the list of instance links

For more information on the different ways to modify the instance link properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying instance link display preferences

You can modify the following display preferences for instance links using the Tools→Display Preferences command.

Preference	When selected
Show association name	Displays the name of the association that the instance link refers to
Show stereotype	Displays the stereotype of the instance link
Show association role names	Displays the names of the association roles

Defining messages in a collaboration diagram

Objects can cooperate by using several kinds of requests (send a signal, invoke an operation, create a new object, delete an existing object, etc…). Sending a signal is used to trigger a reaction from the receiver in an asynchronous way and without a reply. Invoking an operation will apply an operation to an object in a synchronous or asynchronous mode, and may require a reply from the receiver. All these requests constitute **messages**. They correspond to stimulus in the UML language.

The message shares the same concept in the sequence and collaboration diagrams. It can either be created in the diagram type you need, or dragged from a diagram type and dropped into another diagram type.

For more information on messages in the sequence diagram, see section Defining messages in a sequence diagram, in chapter Building a sequence diagram.

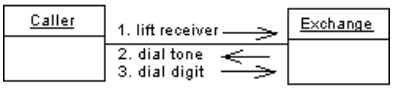
How to draw messages?

The message symbol is an arrow that displays the following information:

- The direction of the message
- The sequence number
- The message, or the operation name
- The condition
- The argument

A message can be drawn horizontally or vertically in the diagram.

Several messages may be attached to the same instance link to specify different interactions. Each message in a collaboration diagram is attached to only one instance link, so the destruction of an instance link destroys all the messages associated with the instance link.



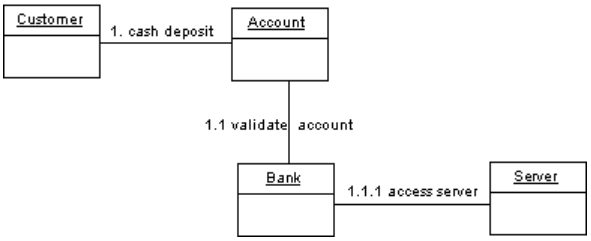
You can create a recursive message: it is a message with the same object as sender and receiver.

Using sequence numbers

A sequence number is appended to a message, it indicates the order in which messages are exchanged. It is the element that defines the order of messages within a collaboration diagram. Sequence numbers are automatically generated if you create the message using the creation tools. They can be manually appended in the Sequence Number box from the message property sheet.

The order of interactions specified by sequence numbers, usually starts with number 1. It is automatically incremented by 1 after each new creation. When you need to change a sequence number manually, you can select the message in the diagram to increase or decrease the sequence number.

You can use sub-numbers like 1.1, 1.1.1, etc ... when you want to describe a nested procedure, or a detailed decomposition of messages exchanged between objects.



If you reuse messages between sequence and collaboration diagrams, the sequence numbers must be manually reordered.

For more information on the sequence numbers order, see section Moving sequence numbers in a collaboration diagram.

Messages behavior

The following rules apply to messages:

- A message cannot be used by two instance links
- Two messages can be defined between the same source and destination objects (**parallel messages**)
- You can use messages in both directions between a sender and a receiver
- When a message is dragged from a sequence diagram to a collaboration diagram, it is dropped with both extremities if they do not exist, and it is attached to a default instance link. The default instance link is created if it does not exist
- When the extremities of the instance link change, the extremities of the message change, but when the message introduces strong constraints such as Create, or Destroy, then these constraints are removed

- It is not possible to invert the direction of a message when the message is a Create or Destroy message

For more information on Create and Destroy messages, see section Detail page of the message in a sequence diagram, in chapter Building a sequence diagram.

For more information on messages, see section Instance links behavior in Defining instance links in a collaboration diagram.

Using a message in several collaboration diagrams

It is possible to use the same message in more than one collaboration diagram. This is valid using a Reuse of the message from one collaboration diagram to another. You are not allowed to use external shortcuts of messages between collaboration diagrams.

The sequence number attached to a message is identical in all diagrams if the message is reused.

Defining message properties in a collaboration diagram

A message has general properties:

Property	Description	Maximum length
Name	Name of the message	254
Code	Reference name or code for the message	254
Comment	Descriptive comment for the message	—
Stereotype	Extends the semantics of a message derived from existing messages but specific to your needs	254
Sequence number	Sequence number of the message	—
Sender	Object the message starts from. You can click the Properties tool beside this box to display the object property sheet	254
Receiver	Object the message ends on. You can click the Properties tool beside this box to display the object property sheet	254
Reverse direction	Reverses both objects at each end of the message	—

Inverting the direction of a message
You can right-click a message in the diagram and select Reverse from the contextual menu to invert its direction.

Detail page of the message in a collaboration diagram

Action

The Detail page in the message property sheet contains an Action dropdown listbox. The values in the Action dropdown listbox allow you to show what characteristics a message can have, it is used to document the properties of the message.

Action	Description
Create	<optional>. Allows you to specify the creation of an object. The Create action means that a specific action instantiates and initializes the receiver object
Destroy	<optional>. Allows you to specify the destruction of an object. The Destroy action means that the sender object destroys the receiver object.
Self-destroy	<optional>. Allows you to specify the self-destruction of an object. It is available when you select the Return control flow type beforehand

Operation

The Detail page in the message property sheet contains an Operation dropdown listbox to link the message to an operation of a class.

If the receiver of a message is an object, and the object has a class, the message, as a dynamic flow of information, invokes an operation. You can therefore link a message to an operation of the class and replace the message name with the name of a method that one object is asking the other to invoke. This process can be very useful during implementation.

You can link a message to an operation, or create a new operation from the Operation dropdown listbox in the Detail page of the message property sheet. If an operation is linked to a message, you can display the name of the operation instead of the name of the message by selecting the Replace by Operation Name display preference in the message category.

You can also define arguments of the operation defined in the message in the Arguments box. This property is available only when an operation is defined within a message, otherwise it is empty and cannot be edited.

For more information on linking a message to an operation, see section Linking a message to an operation, in chapter Working with Object-Oriented models.

Condition, time and support delay

The Detail page also contains the following message options:

- Condition
- Begin time
- End time
- Support delay

Option	Description	Example
Condition	May be specified by placing Boolean expressions in braces on the diagram	Condition for timing: [dialing time < 30 sec]
Begin time End time	User defined time alias, used for defining constraints	Begin time=t1, End time=t2, constraint={t2 - t1 < 30 sec}
Support delay	Defines a transmission delay. If selected, implies that the message may have duration. If deselected, implies that the message is instantaneous, or fast	A Support delay deselected forces the end time to have the same value as the begin time

Accessing the Detail page
You can access the Detail page of a message by right clicking the message symbol in the diagram, and selecting Detail from the contextual menu.

Creating a message in a collaboration diagram

There are several ways to create a message:

- From a diagram
- From the Browser

At creation, a message has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a message, see section Messages page of the instance link.

❖ **To create a message from a collaboration diagram:**

1 Click the Message tool in the palette.

2 Click the sender and while continuing to hold down the mouse button, drag the cursor to the receiver. Release the mouse button on the receiver.
or
Click on the instance link symbol to which you want to attach the message.

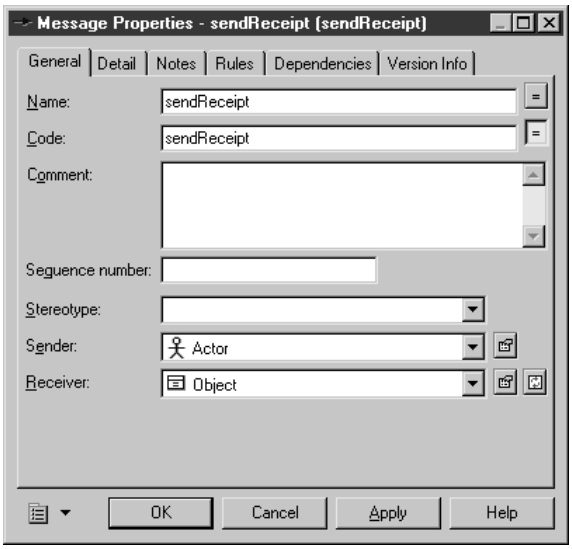
When you create a message and there are several instance links between both objects, the message is attached to the first created instance link.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the tool.

4 Double-click the message symbol to display its property sheet.

5 Type a name and a code.



6 Click the Detail tab to display the Detail page.

7 Select properties as required.

8 Click OK.

Creating a message from the list of messages
When you create a message from the list of messages, an instance link is created between both extremities of the message after you commit the message creation. The instance link is displayed in the Owner Link column.

Creating a sequence number in a collaboration diagram

When you create a message using the Message tool or the other tools, the default value of the sequence number is calculated with regards to the most recently created or modified message. The first sequence number created is 1.

A succession of sequence numbers is built from the most recent sequence number plus 1. For example, 3 + 1 = 4, or 2.1 + 1 = 2.2

The creation of sequence numbers respects the syntax of numbers already used in the diagram (1, 2, 3, etc... or 1.1, 1.2, etc...).

Managing sequence numbers in a collaboration diagram

If you need to change sequence numbers manually, you can move or insert messages in the diagram or increase and decrease the sequence numbers.

Moving sequence numbers in a collaboration diagram

You can move and insert message numbers within the collaboration diagram.

When you move an existing number and attach it to another message, the sequence numbers are recalculated with respect to the following rules:

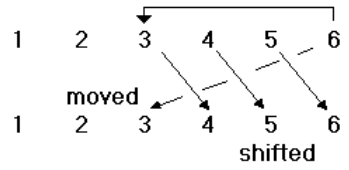
- For a number 'x', all numbers equal to or greater than number 'x' are modified
- Any gap is filled with the sequence number that is immediately available after the move

Here are some examples to help you understand what happens:

Example 1

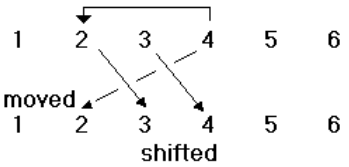
The sequence numbers in a collaboration diagram are 1, 2, 3, 4, 5, and 6.

When you change sequence number 6 and place it in third position, sequence number 6 becomes sequence number 3: all numbers between 3 and 6 are modified as follows:



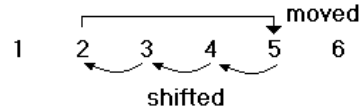
Example 2

When you change sequence number 4 and place it in second position, sequence number 4 becomes sequence number 2: all numbers between 2 and 4 are modified, 5 and 6 remain:



Example 3

When you change sequence number 2 and place it in fifth position, sequence number 2 becomes sequence number 5: all numbers between 2 and 5 are modified as follows:

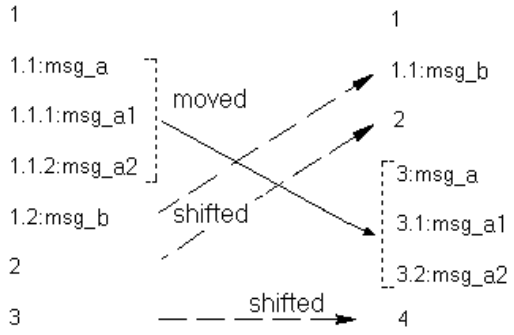


Example 4

The sequence numbers in a collaboration diagram are:

1
1.1:msg_a
1.1.1:msg_a1
1.1.2:msg_a2
1.2:msg_b
2
3

When you change sequence number 1.1:msg_a to sequence number 3, the following changes occur:



Using the Undo feature
You can use the Undo feature whenever needed while manipulating these elements within collaboration diagrams.

Inserting sequence numbers in a collaboration diagram

When you insert a new message with a new sequence number in an existing collaboration diagram, the sequence numbers are recalculated with respect to the following rule: for each number after the insertion, all numbers are incremented by 1.

In the same manner, a parent changes its children. For example, the number is incremented by 1 for numbers like 1.1, 1.2, 1.3 as follows: 1.1 + 1 = 1.2.

The syntax of sequence numbers currently used in the diagram is respected: that means that the number is incremented by 1 regardless of the syntax (1, 2, 3... or 1.1, 1.2, 1.3...).

Increasing sequence numbers in a collaboration diagram

You can increment a sequence number using the following methods:

- Right-click the message in the diagram and select Increase Number from the contextual menu.
  or
  Select the message in the diagram and press CTRL + press the numpad plus sign (+) to increment the number by 1.

Decreasing sequence numbers in a collaboration diagram

You can decrease a sequence number using the following methods:

- Right-click the message in the diagram and select Decrease Number from the contextual menu.
  or
  Select the sequence number in the diagram and press CTRL + press the numpad minus sign (-) to decrease the number by 1.

Modifying message properties in a collaboration diagram

You can modify the message properties using the following methods:

- From the message property sheet
- From the List of Messages

For more information on the different ways to modify the message properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying message display preferences in a collaboration diagram

You can modify the following display preferences for messages using the Tools→Display Preferences command:

Preference	Description
Show name	Displays the name of the message
Replace by operation name	If the message is linked to an operation, the name of the operation replaces the name of the message
Show condition	Displays the condition of the message

CHAPTER 6

Building a Sequence Diagram

About this chapter

This chapter describes how to build a sequence diagram. It explains the role of each element in a sequence diagram and how to create and modify them.

Contents

Topic	Page
Sequence diagram basics	224
Defining actors in a sequence diagram	227
Defining objects in a sequence diagram	231
Defining messages in a sequence diagram	236

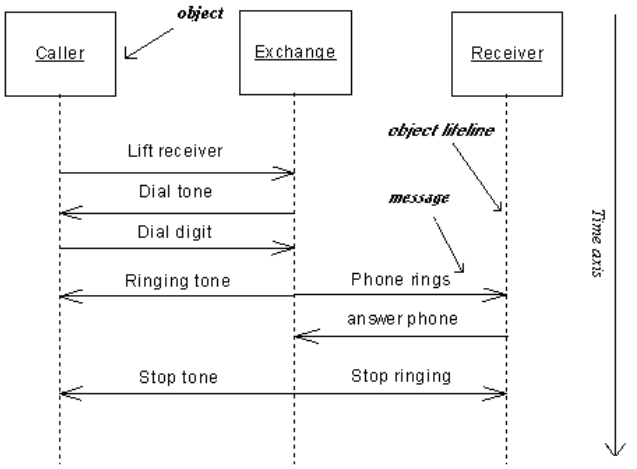
Sequence diagram basics

The **sequence diagram** is part of the Unified Modeling Language (UML). It is a graphical description of the operations of a system based on chronology. It represents the life of objects over a definite period of time.

Defining a sequence diagram

The sequence diagram is part of an Object-Oriented Model. It is one of the available diagrams in an OOM. It is a dynamic view that shows the symbols of objects (instances of a class), and messages passed along. You can also include actors that interact with the system.

It displays an interaction as a two-dimensional chart. The vertical dimension is the time axis, and the horizontal dimension shows the objects roles that represent individual objects in the collaboration.



Sequence diagrams convey the same information as collaboration diagrams, they represent two different views of the same context, the main difference is that a sequence diagram shows objects interactions based on chronology.

For more information on the collaboration diagram, see chapter Building a collaboration diagram.

Why build a sequence diagram?

A sequence diagram represents behavior in terms of interactions. It complements the class diagram that represents the static structure of the system. It specifies the behavior of classes, interfaces, and the possible use of their operations.

Building a sequence diagram can also be the opportunity to test the static model at the conception level. The sequence diagram may indeed represent a scenario in which classes from the class diagram are instantiated to create the objects necessary to run the scenario.

Applicability

Sequence diagrams are particularly important for designers because they clarify the roles of objects in a chronological flow, based on events.

The conception of a model usually starts with building a use case diagram, then build a class diagram with the classes identified in the use case diagram. One or more sequence diagrams may then illustrate the object interactions that enact a use case. A typical organization is to have one sequence diagram for the main flow of events and one sequence diagram for each independent sub-flow of the use case.

For more information on diagrams of the OOM, see section What is an OOM, in chapter Object-Oriented Model Basics.

Creating a sequence diagram

You can create a sequence diagram using different methods.

For more information on creating a sequence diagram in a new OOM, see section Creating an OOM, in chapter Object-Oriented Model Basics.

For more information on creating a sequence diagram in an existing OOM, see section Creating a new diagram, in chapter Managing Models, in the *General Features Guide*.

Symbol menu not available
The Symbol→Group Symbols feature is only available in the sequence diagram for free symbols. Moreover, the Auto-layout, Align and Disposition features are not available in the sequence diagram.

Merging sequence diagrams
When you use the Merge Models feature to merge sequence diagrams, the symbols of all elements in the sequence diagram are merged without comparison. You can either accept all modifications on all symbols or no modification at all.

Converting a sequence diagram to a collaboration diagram

You can convert a sequence diagram to a collaboration diagram.

To do so, create a collaboration diagram at the same level as the existing sequence diagram (under the same package), right-click the diagram background and select Diagram→Show Symbols from the contextual menu. This opens a selection dialog box from which you can select objects of the sequence diagram that will be used into the collaboration diagram.

You can also use a Visual Basic script that converts the existing sequence diagram to a collaboration diagram. To do so, run the script named SequenceToCollaboration.VBS from the sequence diagram by selecting Tools→Execute Commands→Edit/Run scripts. The script is located in the VB Scripts folder of the PowerDesigner installation.

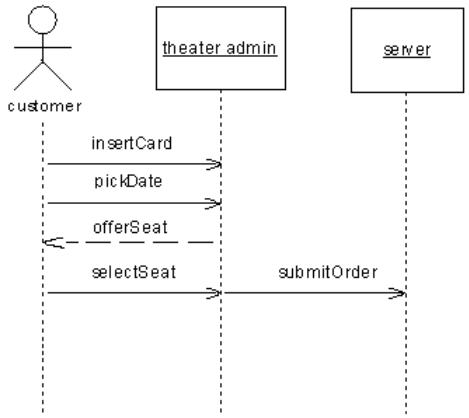
Defining actors in a sequence diagram

The system you are describing interacts with **actors**. The definition of an actor is the same as in a use case diagram or a collaboration diagram, it characterizes an outside user, or a related set of users that interact with a system.

In the sequence diagram, an actor has a lifeline representing the duration of its life. You cannot separate an actor and its lifeline.

If an actor is the invoker of the interaction, it is usually represented by the first (farthest left) lifeline in the sequence diagram. If you have several actors in the diagram, you should try to position them to the farthest left or to the farthest right lifelines because actors are, by definition, external to the system.

Example



Actors can be shared between use case, sequence, and collaboration diagrams if they play the same role. Actors are common to the model. They can either be created in the diagram type you need, or dragged from a diagram type and dropped into another diagram type.

For more information on actors in the use case diagram, see section Defining actors in a use case diagram, in chapter Building a use case diagram.

For more information on actors in the collaboration diagram, see section Defining actors in a collaboration diagram, in chapter Building a collaboration diagram.

Actor properties in a sequence diagram

You can double-click any actor symbol in a diagram to display its properties.

An actor has the following general properties:

Property	Description	Maximum length
Name	Name of the actor	254
Code	Reference name or code for the actor	254
Comment	Descriptive comment for the actor	—
Stereotype	Extends the semantics of an actor derived from existing actors but specific to your needs	254

Actor implementation classes

The Implementation Classes page shows the list of classes and interfaces used to implement an actor.

Accessing the Implementation Classes page
You access the Implementation Classes page of an actor by right clicking the actor symbol in the diagram, and selecting Implementation Classes from the contextual menu.

For more information on implementation classes, see section Linking a class or interface to an actor, in chapter Working with Object-Oriented models.

Actor related diagrams

In the actor property sheet, you can select the Related Diagrams tab to display all diagrams of the model that are related to an actor. They can be attached and detached from the current actor at this stage. This page displays a list of the diagrams identified by their name and code.

For more information on Related Diagrams, see section Defining related diagrams, in chapter Working with Object-Oriented models.

Creating an actor in a sequence diagram

There are several ways to create an actor:

- From a diagram
- From the Browser
- From the List of Actors

At creation, an actor has a default name including a number, this number is assigned in the order of creation.

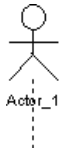
For more information on the different ways to create an actor, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an actor from a sequence diagram:**

1 Click the Actor tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



3 Click the Pointer tool.
or
Click the right mouse button.

You release the Actor tool.

4 Double-click the actor symbol to display its property sheet.

5 Type a name and a code.

6 Click OK.

The actor node appears in the Actors category of the Browser.

Showing the symbol of an actor

You can show the symbol of an actor already created in a use case, or collaboration diagram using different methods.

For more information on how to show an actor symbol, see section Showing the symbol of an actor, in chapter Building a use case diagram.

Modifying actor properties in a sequence diagram

You can modify the actor properties using the following methods:

- From the actor property sheet
- From the List of Actors

For more information on the different ways to modify the actor properties, see section Modifying object properties, in chapter Managing objects in the *General Features Guide*.

Modifying actor display preferences

You can modify the following display preferences for actors using the Tools→Display Preferences command.

Preference	Description
Show stereotype	Displays the stereotype of the actor

Defining objects in a sequence diagram

An **object** is an instance of a class. It shares the same concept in the object, collaboration and sequence diagrams. It can either be created in the diagram type you need, or dragged from a diagram type and dropped into another diagram type.

For more information on objects in the object diagram, see section Defining objects in an object diagram, in chapter Building an object diagram.

For more information on objects in the collaboration diagram, see section Defining objects in a collaboration diagram, in chapter Building a collaboration diagram.

Object lifeline

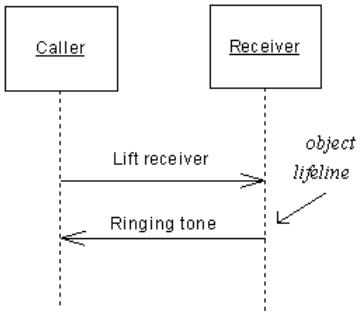
In the sequence diagram, an object has a **lifeline**: it is the dashed vertical line under the object symbol. Time always proceeds down the page. The object lifeline indicates the period during which an object exists. You cannot separate an object and its lifeline.

If the object is created or destroyed during the period of time shown on the diagram, then its lifeline starts or stops at the corresponding point.

Graphical interpretation

Objects appear at the top of the diagram. They exchange messages between them.

An object that exists when a transaction, or message starts, is shown at the top of the diagram, above the first message arrow. The lifeline of an object that still exists when the transaction is over, continues beyond the final message arrow.



Object properties in a sequence diagram

You can double-click any object symbol in a diagram to display its properties.

An object has general properties:

Property	Description	Maximum length
Name	Name of the object. Not compulsory	254
Code	Reference name or code for the object. Not compulsory	254
Comment	Descriptive comment for the object	—
Stereotype	Extends the semantics of an object derived from existing objects but specific to your needs	254
Class	Class from which an object is an instance	254

Name and code

The name and the code are not compulsory since you can have an object representing an unnamed instance of a class, but if the name or code boxes are empty, you need to supply the class name in the Class box.

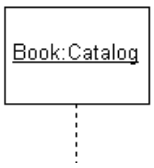
The name and the code should be unique per class. Two objects could share the same name only if they belong to different classes.

Class property of the object

You can link an object to a class, or create a new class from the Class dropdown listbox in the object property sheet.

When you select a class from the object property sheet, the syntax of the object symbol shows the name of this class.

For example, an object instance originally named Book will automatically be renamed Book:Catalog in the diagram if it is an instance of the class Catalog.



You can also drag a class node from the Browser, or drag a class from a class diagram and drop it into the sequence diagram. This will automatically create an object that is an instance of the class.

For more information on linking a class to an object, see section Linking a class to an object, in chapter Working with Object-Oriented models.

Creating an object in a sequence diagram

There are several ways to create an object:

- From a diagram
- From the Browser
- From the List of Objects

At creation, an object has a default name including a number, this number is assigned in the order of creation.

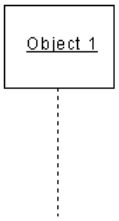
For more information on the different ways to create an object, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an object from a sequence diagram:**

1 Click the Object tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



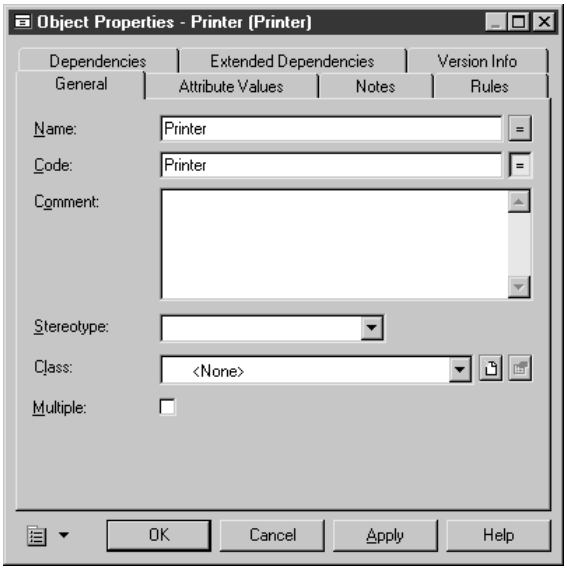
3 Click the Pointer tool.
or
Click the right mouse button.

You release the Object tool.

4 Double-click the object symbol to display its property sheet.

Opening property sheets at last accessed page
Property sheets open to the General page by default. However, you can choose to open property sheets at the last page accessed by selecting Tools→General Options→Dialog, and selecting the option Keep Last Tab in the Property Sheets groupbox.

5 Type a name and a code.



6 Click OK.

The object node appears in the Objects category of the Browser.

Modifying object properties in a sequence diagram

You can modify the object properties using the following methods:

- From the object property sheet
- From the List of Objects

For more information on the different ways to modify the object properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying object display preferences in a sequence diagram

You can modify the following display preferences for objects using the Tools→Display Preferences command.

Preference	Description
Show stereotype	Displays the stereotype of the object

Defining messages in a sequence diagram

A **message** is a communication between objects that conveys information with the expectation that activity will ensue. The receipt of a message should normally have an outcome. It corresponds to a stimulus in UML.

A message has a sender, a receiver, and an action. The **sender** is the object or actor that sends the message. The **receiver** is the object or actor that receives the message. The action is executed to transmit the information.

A message is shown as a horizontal solid arrow from the lifeline of one object to the lifeline of another. The arrow is labeled with the name of the message. You can also define a **control flow type** that represents both the relationship between an action and its preceding and succeeding actions, and the waiting semantics between them.

The message shares the same concept in the collaboration and sequence diagrams. It can either be created in the diagram type you need, or dragged from a diagram type and dropped into another diagram type.

For more information on messages in the collaboration diagram, see section Defining messages in a collaboration diagram, in chapter Building a collaboration diagram.

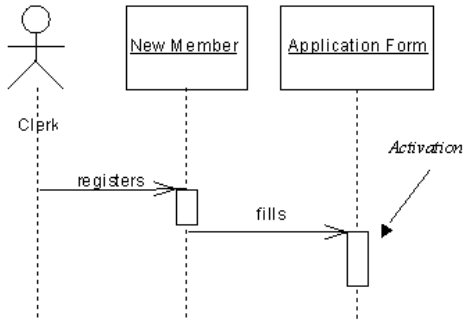
Activation

When you create a message, you can choose to create an activation at the same time. Drawing an activation can be particularly useful if you want to describe a procedural flow for example.

An **activation** represents the period of time during which an object is performing an action. It only exists as a graphic symbol without a property sheet; there is no Activations node in the Browser, and no list of activations.

An activation is shown as a thin rectangle. The top of an activation is aligned with the initiation time and the bottom of an activation is aligned with the completion time. You can add a note next to the activation symbol if you need to explain more about the role of the activation.

Example



For more information on activations, see section Creating an activation.

A message can be drawn from an actor to an object, or inversely. It is also possible to create a message between two actors but it will be detected, and displayed as a warning during the check model process.

A recursive message is a message sent from an object to itself, the sender and the receiver are the same object.

Using a message in several sequence diagrams

It is possible to use the same message in more than one sequence diagram. This is valid using a copy of the message from one sequence diagram to another. You are not allowed however to use external shortcuts of messages between sequence diagrams.

When you copy a message, its name does not change. You can either keep its original name, or rename the message after copy.

Any change on the Action or Control Flow value of the message is reflected in all diagrams. However, if the change you want to perform is not valid, the change will not be possible. For example, you are not allowed to move a Create message if a Create message already exists between the sender and the receiver.

The sequence number attached to a message is identical in all diagrams if the message is reused.

Defining message properties in a sequence diagram

A message has general properties:

Property	Description	Maximum length
Name	Name of the message	254
Code	Reference name or code for the message	254
Comment	Descriptive comment for the message	—
Stereotype	Extends the semantics of a message, derived from existing messages but specific to your needs	254
Sender	Object the message starts from. You can click the Properties tool beside this box to display the object property sheet	254
Receiver	Object the message ends on. You can click the Properties tool beside this box to display the object property sheet	254
Sequence number	Allows you to manually add a sequence number to the message. It is mainly used in collaboration diagrams but can also be used in sequence diagrams	—
Reverse direction	Inverts both objects at each end of the message	—

Detail page of the message in a sequence diagram

The Detail page in the message property sheet applies to messages created in sequence and collaboration diagrams.

For more information on messages in collaboration diagrams, see section Detail page of the message in a collaboration diagram, in chapter Building a collaboration diagram.

The Detail page in the message property sheet contains an Action dropdown listbox.

Create action

The Create action allows you to show the creation of an object in a sequence diagram. It means that a specific action instantiates and initializes the receiver object. A message with a create action is the first message between a sender and a receiver.

For more information on the create action, see section Creating a Create message in a sequence diagram.

Destroy action

The Destroy action allows you to show the destruction of an object in a sequence diagram. It means that the sender object destroys the receiver object. A large X is displayed on the lifeline of the receiver object. A message with a destroy action is the last message between a sender and a receiver.

For more information on the destroy action, see section Creating a Destroy message in a sequence diagram.

Self-Destroy action

The Self-Destroy action allows you to show the self destruction of an object in a sequence diagram. It means that the sender object warns the receiver object that it is destroying itself. A large X is displayed on the lifeline of the sender object. A message with a self-destroy action is the last message between a sender and a receiver.

It is only available if you select the Return control flow from the Control Flow dropdown listbox in the message property sheet.

For more information on the self-destroy action, see section Creating a Self-Destroy message in a sequence diagram.

Control flow

The Control Flow dropdown listbox defines the different types of message control flow.

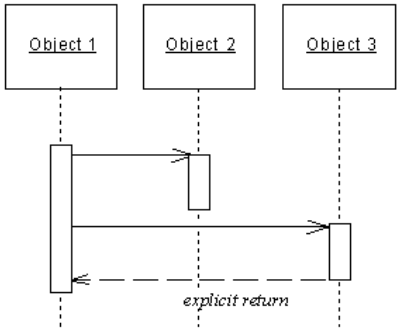
A message has the following control flow values:

Control flow	Description	Symbol
Asynchronous	Request in which the sending object does not wait for a result, it can do something else in parallel. **No-wait semantics**	
Procedure Call	Call of a procedure. The sequence is complete before the next sequence resumes. The sender must wait for a response or the end of the activation **Wait semantics**	
Return	Generally associated with a Procedure Call. The Return arrow may be omitted as it is implicit at the end of an activation	
Undefined	No control flow defined	

By default, a message has an Undefined control flow.

If you want to make a diagram more readable, you can draw the Return arrow to show the exact time when the action is returned back to the sender. It is an explicit return that results in returning a value to its origin.

In the example below, the explicit return causes values to be passed back to the original activation.



Control flow vs actions

You can combine message control flows and message actions according to the following table:

Control flow	No action	Create	Destroy	Self-Destroy
Asynchronous	✔	✔	✔	—
Procedure Call	✔	✔	✔	—
Return	✔	—	—	✔
Undefined	✔	✔	✔	—

✔ = allowed

— = not allowed

Accessing Action and Control flow values
You can access the Action and Control flow values of a message by right clicking the message symbol in the diagram, and selecting Action/Control flow from the contextual menu.

Operation

The Detail page in the message property sheet contains an Operation dropdown listbox to link the message to an operation of a class.

If the receiver of a message is an object, and the object has a class, the message, as a dynamic flow of information, invokes an operation.

You can therefore link a message to an operation of the class and replace the message name with the name of a method that one object is asking the other to invoke. This process can be very useful during implementation.

You can link a message to an operation, or create a new operation from the Operation dropdown listbox in the Detail page of the message property sheet. If an operation is linked to a message, you can display the name of the operation instead of the name of the message by selecting the Replace by Operation Name display preference in the message category.

You may even link a Create message to a Constructor operation of a class if you wish to further detail a relation between a message and an operation.

You are not allowed to link a message with a Return control flow to an operation.

You can also define arguments of the operation defined in the message in the Arguments box. This property is available only when an operation is defined within a message, otherwise it is empty and cannot be edited.

For more information on linking a message to an operation, see section Linking a message to an operation, in chapter Working with Object-Oriented models.

Condition and time

The Detail page contains the following other message options:

- Condition
- Begin time
- End time
- Support delay

Option	Description	Example
Condition	May be specified by placing Boolean expressions in braces on the diagram	Condition for timing: [dialing time < 30 sec]
Begin time End time	User defined time alias, used for defining constraints	Begin time=t1, End time=t2, constraint={t2 - t1 < 30 sec}
Support delay	Defines a transmission delay	A Support delay deselected forces the end time to have the same value as the begin time

Support delay

If support delay is selected, it implies that the message may have duration, the message symbol will be slanted, and always oriented towards the bottom. If support delay is deselected, it implies that the message is instantaneous, or fast, the message symbol will always be horizontal in the diagram.

You can also select Support delay as a default option in the Model Options dialog box.

Support delay is not available with a recursive message: it is selected and grayed out.

Accessing the Detail page
You can access the Detail page of a message by right clicking the message symbol in the diagram, and selecting Detail from the contextual menu.

Creating a message in a sequence diagram

There are several ways to create a message:

- From a diagram
- From the Browser

At creation, a message has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a message, see section Creating an object in chapter Managing objects in the *General Features Guide*.

You can create messages from the tool palette. When you create Undefined or Return messages from the tool palette, the control flow value is already selected:

Message type	Tool
Undefined message	
Return message	

You can also create an Undefined message and change the control flow value afterwards.

❖ **To create a message from a sequence diagram:**

1 Click the Message or the Return Message tool from the palette.

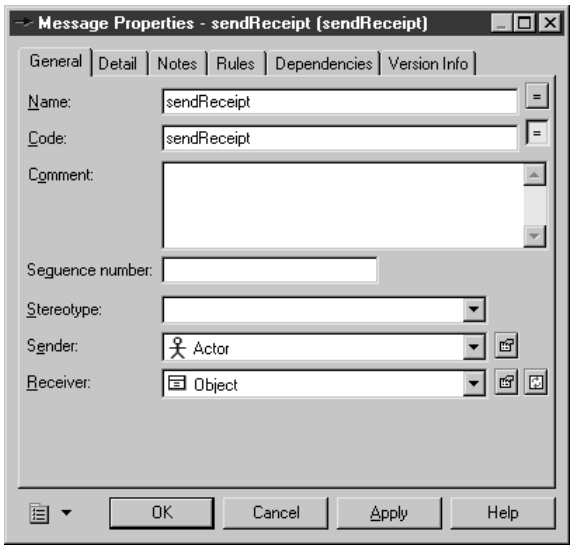
2 Click the lifeline of the sender and while continuing to hold down the mouse button, drag the cursor to the lifeline of the receiver. Release the mouse button on the lifeline of the receiver.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the tool.

4 Double-click the message symbol to display its property sheet.

5 Type a name and a code.



6 Click the Detail tab to display the Detail page.

7 Check the control flow value from the Control Flow dropdown listbox.

8 Click OK.

The message node appears in the Messages category of the Browser.

Using a note and a link
If you need to fully describe, or put a label on a message, you can write a note using the Note tool, and position the note close to the message.

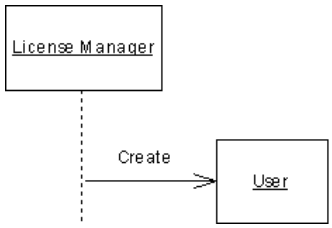
Creating a Create message in a sequence diagram

A message can create an object if you select the Create action in the message property sheet.

For more information on the Create action, see section Detail page of the message in a sequence diagram.

Create is not allowed if the receiver is an actor. You are not allowed either to use Create in a recursive message.

When a message creates an object, the message is drawn with its arrowhead on the object; both object and message are at the same level.



❖ **To create a Create message:**

1 Click the Object tool and create a sender and a receiver objects.

2 Click the Message tool from the palette.

3 Click the lifeline of the sender and while continuing to hold down the mouse button, drag the cursor diagonally to the object symbol of the receiver. Release the mouse button on the object symbol of the receiver.

The object symbol of the receiver moves down to line up with the message arrow.

4 Double-click the newly created message to display the message property sheet.

5 Check that Create is selected from the Action dropdown listbox in the Detail page.

6 Click OK.

Creating Destroy messages in a sequence diagram

A message can destroy an object if you select the Destroy action.

For more information on the Destroy action, see section Detail page of the message in a sequence diagram.

A destroyed object lifeline is marked by a large X in the diagram.

Caution

The Destroy action does not destroy the object, it only represents the destruction of the object in the diagram. The destruction is not visible from the Browser. The object lifeline ends at a precise point in time; it is not possible to graphically pull the lifeline downwards any more.

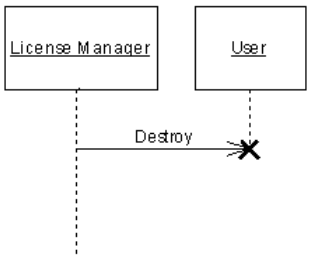
There are two possible destructions:

- Create a Destroy message
- Create a Self-Destroy message

Creating a Destroy message in a sequence diagram

In case of destruction, a large X is displayed at the intersection point between the object lifeline and the message. The Destroy action ends both activation and object lifeline at this same point.

Destroy is not allowed if the receiver is an actor. You are not allowed either to use the Destroy action to show destruction of a recursive message.



❖ **To create a Destroy message:**

1. Click the Object tool and create a sender and a receiver objects.
2. Click the Message tool from the palette.
3. Click the lifeline of the sender and while continuing to hold down the mouse button, drag the cursor to the lifeline of the receiver. Release the mouse button on the lifeline of the receiver.
4. Double-click the newly created message symbol to display the message property sheet.
5. Select Destroy from the Action dropdown listbox in the Detail page.
6. Click OK.

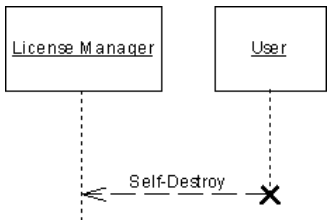
A large X, at its intersection point with the Destroy message arrow marks the object lifeline.

Creating a Self-Destroy message in a sequence diagram

The Self-Destroy action is allowed only when you choose a Return control flow.

If the object is self-destroyed in the diagram, its destruction is marked by a large X on the object lifeline. This represents the end of the sender object.

Self-Destroy is not allowed with an actor. You are not allowed either to use Self-Destroy in a recursive message.



❖ **To create a Self-Destroy message:**

1 Click the Object tool and create a sender and a receiver object.

2 Click the Message tool from the palette.

3 Click the lifeline of the sender and while continuing to hold down the mouse button, drag the cursor to the lifeline of the receiver. Release the mouse button on the lifeline of the receiver.

4 Double-click the newly created message symbol to display the message property sheet.

5 Select Return from the Control flow dropdown listbox in the Detail page.

6 Select Self-Destroy from the Action dropdown listbox.

7 Click OK.

The lifeline of the self-destroyed object is marked by a large X.

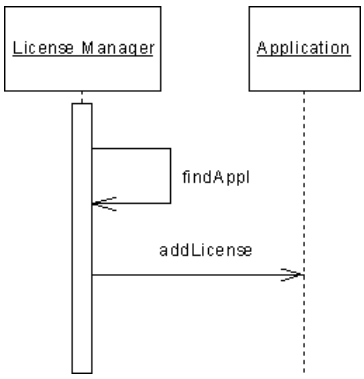
Creating a recursive message in a sequence diagram

A message can be recursive when the object is sending a message to itself. In this case, the arrow starts and finishes on the lifeline of the same object.

You are not allowed to use the Create or the Self-Destroy action with a recursive message.

The Support delay option is not available with a recursive message, it is selected and grayed out.

Example



You can choose to create a recursive message with or without activation from the tool palette.

Creating a recursive message without activation

You can create recursive messages from the tool palette. When you create Undefined or Return recursive messages from the tool palette, the control flow value is already selected:

Message type	Tool
Undefined recursive message	
Return recursive message	

You can also create an Undefined recursive message and change the control flow value afterwards.

❖ **To create a recursive message without activation:**

1 Click the Self Message or the Self Return Message tool from the palette.

2 Click the object lifeline.

The self message symbol appears at the click position.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the tool.

4 Double-click the self message symbol to display the message property sheet.

5 Type a name and a code.

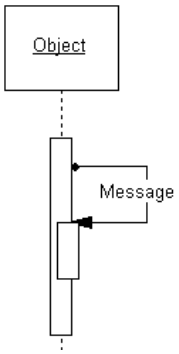
6 Click OK.

The message node appears in the Messages category of the Browser.

Creating a recursive message with activation

When you create a recursive message with activation, the recursive message is automatically attached to an activation and its control flow value is a Procedure Call which, by default, starts the activation.

Activation symbols are then automatically created on the object lifeline as shown below:



❖ **To create a recursive message with activation:**

1 Click the Self Call Message tool from the palette.

2 Click the object lifeline.

The symbol of the self call message appears at the click position.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Self Call Message tool.

4 Double-click the symbol of the self call message to display the message property sheet.

5 Type a name and a code.

6 Click OK.

The message node appears in the Messages category of the Browser.

Creating an activation

Creating activations is optional. You can create them to emphasize the time that is necessary for an action to be performed. It can only be created on the lifeline of an object.

You can choose to either create an activation from a diagram, or to create an activation automatically at creation of the message.

Creating an activation from a diagram

❖ **To create an activation from a diagram:**

1 Click the Activation tool from the palette.

2 Click the lifeline of the object.

The activation symbol appears at the click position.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Activation tool.

The activation symbol is created in the diagram.

If you create the activation on top of existing messages, these ones are automatically attached to the activation.

Creating an activation at creation of the message

Creating an activation at creation of the message applies to Procedure Call messages. A Procedure Call generally starts an activation, that is why the message is automatically attached to an activation.

If you use this procedure, an activation symbol is created on the receiver lifeline as well as on the sender lifeline, if there is none already.

❖ **To create an activation at creation of the message:**

1 Click the Call Message tool from the palette.

2 Click the lifeline of the sender and while continuing to hold down the mouse button, drag the cursor to the lifeline of the receiver. Release the mouse button on the lifeline of the receiver.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Call Message tool.

The activation symbol appears at the click position.

Managing activations

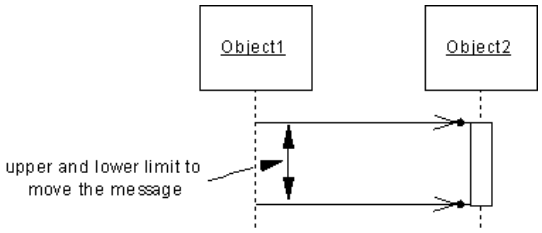
When an activation exists, you can attach or detach a message to it. You can also manipulate the activation through overlap, move, or resize.

Attaching a message to an activation

A message is attached to an activation when its begin or endpoint is on the activation symbol and not on the object lifeline. Attachment symbols are at the endpoints of the message arrow. If attachment symbols are not displayed in the diagram, select Tools→Display Preferences, and select the Show Activation Attachment display preference in the Message category.

You can attach an existing message to an activation by dragging the message inside the activation, while holding the CTRL key down.

When a message is attached to an activation, you cannot move it outside the limits of the activation symbol as shown below:



If you delete an activation with a message attached, the message will be detached from the activation but will not be deleted.

Message control flow and activation

When a message is attached to an activation, the control flow value of the message influences the position of the activation towards the message:

Procedure Call A Procedure Call message attached to an activation starts the activation on the receiver lifeline, that is to say the arrival point of the message is located at the top of the activation.

Return A Return message attached to an activation finishes the activation on the sender lifeline, that is to say the starting point of the message is located at the bottom of the activation.

Procedure Call and Return messages are the only messages defined on a definite location in the activation: a Procedure Call message is at the top of the activation, a Return message is at the bottom of the activation. Other messages attached to an activation can be moved without any constraint inside the activation.

Detaching a message from an activation

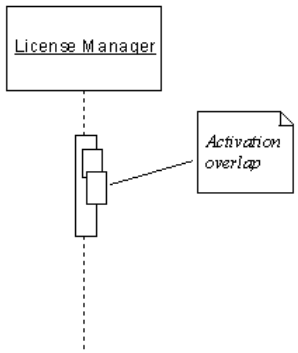
You can detach a message from an activation by dragging the message outside the activation, while holding the CTRL key down.

Overlapping activations

An activation can overlap other existing activations.

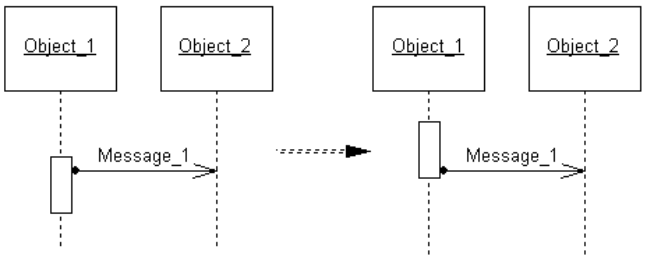
For example, you may want an activation to overlap another one to represent an action in a loop. The action is done repeatedly until it reaches its goal, this loop can start and finish at the same time another activation is representing another action.

You can make activations overlap this way to show concurrent activities.

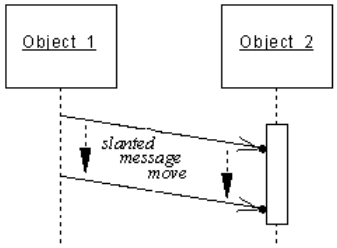


Moving an activation

If you move an activation with a message attached to it, it is possible to move the activation up and down until its top or bottom reaches the message level. The message itself does not move:



If you move the endpoint of a message with Support Delay, the inclination angle of the message is preserved as shown below:



Resizing an activation

You may want to resize an activation by pulling the top or bottom of the activation symbol. This will resize the activation vertically.

When you resize an activation, the following rules apply:

- A Procedure Call message is always attached to the top of the activation on the receiver lifeline. The Procedure Call message stays at the top of the activation if the activation is moved up, or resized upwards.
- A Return message is always attached to the bottom of the activation on the sender lifeline. The Return message stays at the bottom of the activation if the activation is moved down.
- Messages that are covered by the activation after resizing are not automatically attached to the activation.

To change the activation of a message, press CTRL and CLICK to select the begin or endpoint of the message and drag it onto another activation.

Modifying message properties in a sequence diagram

You can modify the message properties using the following methods:

- From the message property sheet
- From the List of Messages

For more information on the different ways to modify the message properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying message display preferences in a sequence diagram

You can modify the following display preferences for messages using the Tools→Display Preferences command:

Preference	Description
Show name	Displays the name of the message
Replace by operation name	If the message is linked to an operation, the name of the operation replaces the name of the message
Show stereotype	Displays the stereotype of the message
Show time	Displays the begin and end times
Show condition	Displays the condition
Show activation attachment	Displays an endpoint at the intersection of the message and the activation

CHAPTER 7

Building a Statechart Diagram

About this chapter

This chapter describes how to build a statechart diagram. It explains the role of each element in a statechart diagram and how to create and modify them.

Contents

Topic	Page
Statechart diagram basics	256
Defining starts in a statechart diagram	261
Defining states	264
Defining actions	272
Defining transitions in a statechart diagram	278
Defining events	285
Defining junction points	291
Defining ends in a statechart diagram	294

Statechart diagram basics

The **statechart diagram** is a diagram that describes a classifier behavior.

Defining a statechart diagram

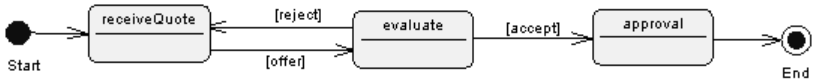
The statechart diagram is one of the diagrams of the Unified Modeling Language (UML). It belongs to the behavioral package in UML (like the use case, collaboration, sequence, and activity diagrams). Its objective is to describe the behavior of a classifier.

Statechart diagram and State machine

The statechart diagram is the graphical representation of a State Machine, a State Machine being a specification that describes the public behavior of some model element that can be a use case, a component or a class. This means that when you start working with a statechart diagram, it is assumed that the classifier has previously been identified in a use case or a class diagram.

The slight difference between a statechart diagram and a State Machine is that the statechart diagram usually represents one instance of a classifier, whereas the State Machine is more global; it represents all instances of the classifier.

The statechart diagram models a finite number of states that the classifier can have, and the events that generate transitions between states.



A statechart diagram can also represent the contents of a composite state, it is called a sub-statechart diagram in this case.

For more information on sub-statechart diagrams, see section Statechart diagrams vs sub-statechart diagrams.

Why build a statechart diagram?

The statechart diagram is built to specify the behavior of an already identified structural element. It specifies a classifier behavior through execution rules explaining precisely how actions are executed during transitions between different states; these states correspond to different situations during the life of the classifier.

As an example, you can build two different statechart diagrams to show:

- What elements are used for a classifier 'Person' with an attribute ‘Age’
- What elements are used for the same classifier with an attribute ‘Marital status’

One application area of the statechart diagram in UML is the Protocol State Machine that specifies a list of operations available on an object, but also the order and conditions into which those operations are invoked. Another potential use is the specification of a Graphic User Interface (GUI) where the states are the distinct screens available with possible transitions between them, all depending on keyboard and mouse events produced by the user.

For more information on diagrams of the OOM, see sections What is an OOM, in chapter Object-Oriented Model Basics.

Statechart diagrams vs sub-statechart diagrams

You can create a statechart diagram within a model or a package, and you can create a sub-statechart diagram within a composite state.

Statechart diagram

You can create several statechart diagrams in a package or a model. Each of those diagrams is independent and designs a specific isolated context in which the integrity of elements can be checked.

In a statechart diagram, it is possible to:

- Create packages
- Convert the statechart diagram to a package

  For more information on converting to a package, see section Converting a diagram to a package, in chapter Managing Models in the *General Features Guide*.

- Convert the statechart diagram to a composite state

  For more information on converting to a composite state, see section Converting a statechart diagram to a composite state.

Sub-statechart diagram

A sub-statechart diagram designs a state decomposition. It details the behavior of a parent state, called composite state. All integrity checks are performed on all elements defined in the composite state.

In a sub-statechart diagram, it is **not** possible to create packages. That is why the package tool appears gray in the tool palette once you are in a sub-statechart diagram. You can only use shortcuts of packages.

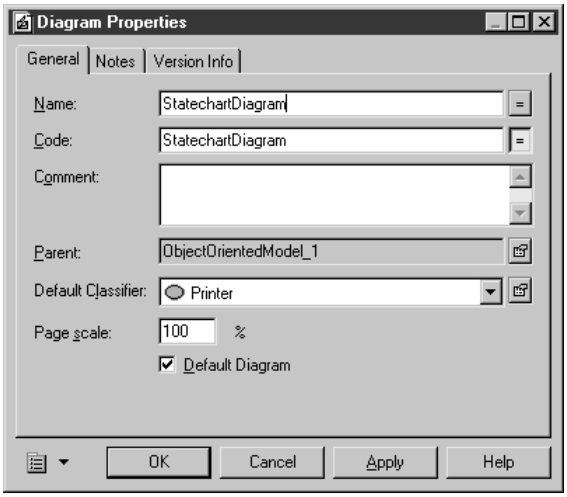
You could create several sub-statechart diagrams within a composite state, however it is generally of no interest to define several sub-statechart diagrams under the same parent state.

For more information on sub-statechart diagrams, see section Defining composite states.

Defining a default classifier in the statechart diagram

You can define the classifier of a state using the Classifier dropdown listbox in the state property sheet. This allows you to link the state to a use case, a component or a class.

At the diagram level, you can also specify the context element of a state by filling in the Default Classifier dropdown listbox in the statechart diagram property sheet. As a result, each state that is created in a diagram using the State tool is automatically associated with the default classifier specified in the statechart diagram property sheet.



By default new diagrams are created with an empty value in the Default Classifier dropdown listbox, except sub-statechart diagrams that automatically share the same Classifier value defined on the parent composite state. The Default Classifier value is an optional value in the statechart diagram.

Creating a statechart diagram

You can create a statechart diagram using different methods.

For more information on creating a statechart diagram in a new OOM, see section Creating an OOM, in chapter Object-Oriented Model Basics.

For more information on creating a statechart diagram in an existing OOM, see section Creating a new diagram, in chapter Managing Models, in the *General Features Guide*.

Converting a statechart diagram to a composite state

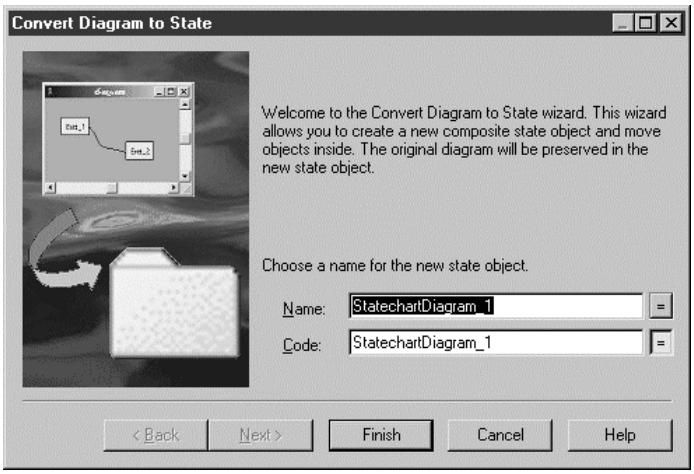
PowerDesigner lets you convert a diagram to a composite state using the Convert to Composite State Wizard from the Tools menu. You do so, when you want to describe the context of a complex state. The diagram must contain objects to proceed to the conversion in the wizard.

When you convert a statechart diagram to a composite state, you convert all objects of the statechart diagram: they are all moved into the new composite state.

❖ **To convert a statechart diagram to a composite state:**

1 Right-click the diagram node in the Browser and select Convert to Composite State from the contextual menu.
or
Right-click the diagram background window and select Diagram→Convert to Composite State from the contextual menu.
or
Select Tools→Convert to Composite State.

The Convert Diagram to State dialog box appears.


Convert Diagram to State
Welcome to the Convert Diagram to State wizard. This wizard allows you to create a new composite state object and move objects inside. The original diagram will be preserved in the new state object.
Choose a name for the new state object.
Name:
StatechartDiagram_1
Code:
StatechartDiagram_1
< Back
Next >
Finish
Cancel
Help

2 Type a name and a code.

If you have created file objects in the statechart diagram, click Next to select the ones you want to move into the composite state.

3 Click Finish.

All objects of the statechart diagram are moved into the composite state. All objects are moved at the same time since it is not possible to create shortcuts for objects of the statechart diagram.

Defining starts in a statechart diagram

A **start** is a starting point of the whole process represented in the statechart diagram. Its symbol is a solid ball as shown below:



The start has the same meaning in the statechart diagram and in the activity diagram, that is why you can use the same start simultaneously in a statechart and in an activity diagram.

How many starts are allowed?

You can create several starts within a package or a model since you may use several statechart diagrams in the same model or package, but only **one** start is allowed in the statechart diagram. As per the composite state, you can only create **one** start in the default sub-statechart diagram.

For this reason, the Start tool is grayed in the palette if a start symbol already exists in the current diagram. Pasting a start is also forbidden if a start already exists in the statechart or the sub-statechart diagram.

You are not allowed to use shortcuts of starts.

Merging models
The start is compared and merged in the Merge Model feature. Since starts should be unique in composite states, the Merge Model feature checks that there is no additional start in a composite state.

Start properties in a statechart diagram

You can double-click any start symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the start	254
Code	Reference name of the start	254
Comment	Descriptive comment for the start	__
Stereotype	Extends the semantics of a start derived from existing starts but specific to your needs	254

Creating a start in a statechart diagram

You create a start from a diagram using the tool palette.

❖ **To create a start from a statechart diagram:**

1 Click the Start tool.

2 Click anywhere in the diagram.

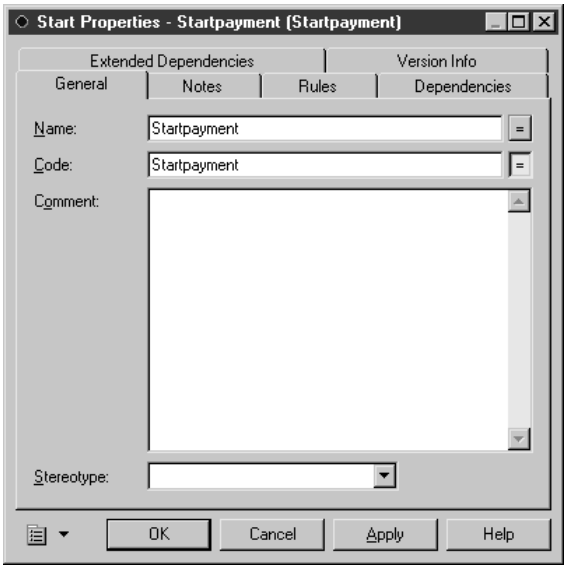
The following symbol appears at the click position:



The Start tool is grayed is the palette after creation since you can only create one start per diagram.

3 Double-click the start symbol to display the start property sheet.

4 Type a name and a code.



5 Click OK.

Modifying start properties in a statechart diagram

You can modify the start properties using the following methods:

- From the start property sheet
- From the list of starts

For more information on the different ways to modify the start properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying start display preferences

You can modify the following display preferences of a start using the Tools→Display Preferences command.

The same display preferences apply to starts and ends in statechart and activity diagrams.

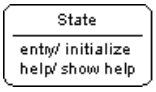
Preference	Description
Show name	Displays the name of the start
Show stereotype	Displays the stereotype of the start

Defining states

A **state** represents a situation during the life of a classifier that is usually specified by conditions. It can also be defined as the situation of a classifier waiting for events. Stability and duration are two characteristics of a state. Several states in a statechart diagram correspond to several situations during the life of the classifier.

Events and condition guards on output transitions define the stability of a state. Some actions can be associated with a state, especially when the object enters or exit the state. Some actions can also be performed when events occur inside the state; those actions are called internal transitions, they do not cause a change of state.

Example



A state can be atomic or composite:

- An atomic state does not contain sub-states
- A composite state uses sub-states to describe its actions

For more information on composite states, see section Defining composite states.

You are not allowed to use shortcuts of states.

Drag a class, use case or component in a statechart diagram

The statechart diagram describes the behavior of a classifier. To highlight the relationship between a classifier and a state, you can define the context classifier of a state using the Classifier dropdown listbox in the state property sheet. This links the state to a use case, a component or a class.

You can also move, copy and paste, or drag a class, use case or component and drop it into a statechart diagram to automatically create a state associated with the element that has been moved.

State properties

You can double-click any state symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the state	254
Code	Reference name of the state	254

Property	Description	Maximum length
Comment	Descriptive comment for the state	__
Stereotype	Sub-classification derived from an existing state. It lets you extend the semantics of an object without changing its structure and can be predefined or user-defined	254
Classifier	Classifier linked to the state. It can be a use case, a class or a component. When a classifier is selected, it is displayed in between brackets after the state name in the Browser	254
Composite	If selected, implies that the state becomes a composite state	__

Actions page of the state

You can specify a set of internal actions on an atomic or composite state from the Actions page in the state property sheet. They represent actions performed within the scope of the state when some events occur. You can create and define the properties of the action from the Actions page, or double-click the arrow at the beginning of a line to display the action property sheet.

Opening the Action page
You can open the Actions page by right clicking the state symbol in the diagram, and selecting Actions from the contextual menu.

For more information on actions, see section Defining actions.

Deferred events page of the state

The Deferred Events page contains an Add Objects tool that allows you to add already existing events but not to create new events. This list is similar to the list of Business Rules that only reuse elements and does not create them.

The difference between an event and a **deferred event** is that an event is always instantaneous and dynamically handled by a state, whereas a deferred event is an event that occurs during a particular state in the object life cycle but it is not directly used up by the state.

A deferred event occurs in a specific state, is then handled in a queue, and is triggered by another state of the same classifier later.

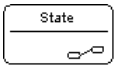
Defining composite states

A state can be composite if you need to refine its behavior. There is no limitation on the number of decompositions. Each state can be decomposed into a composite state, except the lowest level states (they do not need to be further decomposed because they intrinsically contain enough detail).

The Composite check box in the state property sheet allows you to create or delete a composite state as follows:

- If you select the Composite box, a sub-statechart diagram is created within the state to specify the state decomposition. You can then create any objects of the statechart diagram
- If you deselect the Composite box, or select Change to atomic from the contextual menu of a composite state, a confirmation dialog box asks you to confirm the objects deletion. The existing sub-statechart diagram and all its objects are then deleted. There is no confirmation dialog box if no objects have been defined in the sub-statechart diagram

The composite state symbol appears with an icon in the right-hand corner.



A composite state always contains a default sub-statechart diagram. It would be possible to create several sub-statechart diagrams within a composite state, but it is not recommended because being in a sub-statechart diagram means being within the context of a state. Unless you want to design some exception cases like error management for example, it would not be consistent to create too many sub-statechart diagrams within a composite state.

The composite state behaves like a **specialized** package or container that contains a sub-statechart diagram, for this reason, it is not possible to create a package, or any other UML diagram type in a composite state.

Dragging statechart diagram objects into a composite state
You can drag any statechart diagram objects from the Browser and drop them into a sub-statechart diagram. These objects are automatically inserted into the default sub-statechart diagram.

All sub-states defined within a composite state share the same namespace as the parent state.

Including composite states
To display all states in composite states of the model, click the Include Composite States tool in the list of states from the Model menu.

When you work within a composite state, the associated objects appear under the States node in the Browser. The States node can be expanded to display its child nodes, this allows you to see the state decomposition in a quick and convenient way.

Find in Browser
You can locate any object or any diagram in the Browser tree view from the current diagram window. To do so, right-click the object symbol, or the diagram background and select Edit→Find in Browser.

Creating a state

There are several ways to create a state:

- From a diagram
- From the Browser
- From the list of states

At creation, a state has a default name including a number, this number is assigned in the order of creation.

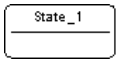
For more information on the different ways to create a state, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a state from a diagram:**

1 Click the State tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



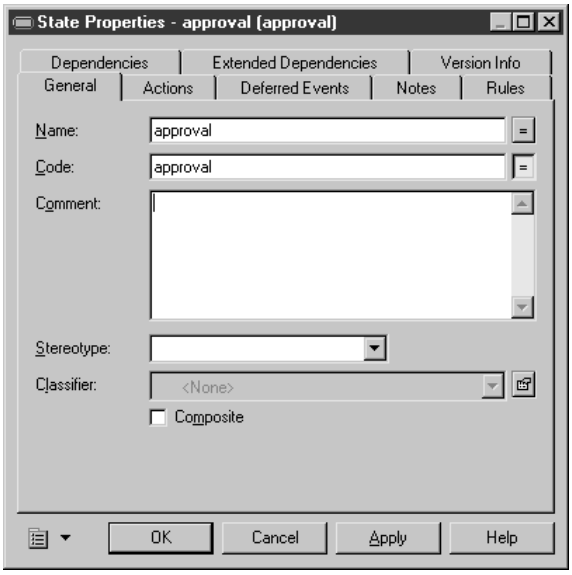
3 Click the Pointer tool.
or
Click the right mouse button.

You release the State tool.

4 Double-click the state symbol to display the state property sheet.

> **Opening property sheets at last accessed page**
> Property sheets open to the General page by default. However, you can choose to open property sheets at the last page accessed by selecting Tools→General Options→Dialog, and selecting the option Keep Last Tab in the Property Sheets groupbox.

5 Type a name and a code.



6 Click OK.

Creating a composite state

You create a composite state from the symbol of the parent state:

- From the diagram
- From the contextual menu

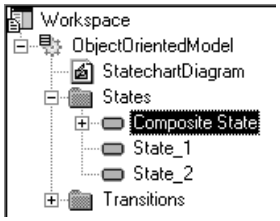
When you create a composite state, a default sub-statechart diagram is automatically created.

Creating a composite state from the diagram

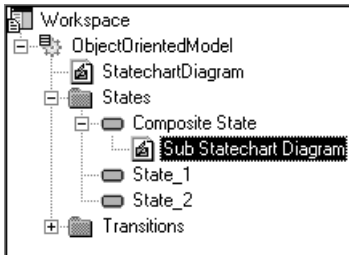
❖ **To create a composite state from the diagram:**

♦ Press CTRL and double-click the parent state symbol to create the composite state

A default sub-statechart diagram opens in the diagram window. The composite state node is visible in the States node of the Browser marked with a + sign.



If you expand the composite state node, you see that a default sub-statechart diagram node is automatically created.



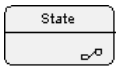
Creating a composite state from the contextual menu

You can create a composite state from the state contextual menu, it is equivalent to selecting the Composite box in the state property sheet.

❖ **To create a composite state from the contextual menu:**

♦ Right-click the parent state symbol and select Change to Composite from the contextual menu.

The state symbol changes to the following symbol:



Opening a composite state

You can open a composite state using the following methods:

- Press CTRL and double-click on the composite state symbol.
  or
  Select Open Diagram from the contextual menu of the composite state.

  The default sub-statechart diagram opens in the diagram window.

Closing a composite state

You can close a composite state using the following methods:

- Press CTRL + U in the sub-statechart diagram to go up to the default diagram.
  or
  Select Diagram→Go Up One Level from the contextual menu.
  If several diagrams are open in the workspace, you are requested to select the parent statechart diagram.

> **Setting a diagram as default diagram**
> You can set a diagram to be the default diagram by selecting the Default Diagram check box in the model property sheet.

Modifying state properties

You can modify the state properties using the following methods:

- From the state property sheet
- From the list of states

For more information on the different ways to modify the state properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying state display preferences

You can modify the following display preference for states using the Tools→Display Preferences command:

Preference	Description
Show actions	Displays the actions associated with the state
Show event arguments	Displays the arguments of the event (defined on actions)
Show stereotype	Displays the stereotype of the state

For more information on event arguments, see section Defining event arguments.

Defining actions

An **action** is a specification of a computable statement. It occurs in a specific situation and may comprise predefined events (entry, do and exit) and internal transitions.

For more information on the entry, do and exit predefined events, see section Defining events.

Internal transitions can be defined on a state, they are internal to the state and do not cause a change of state; they perform actions when triggered by events. Internal transitions should not be compared to reflexive transitions on the state because the entry and exit values are not executed when the internal event occurs.

An action contains a **Trigger Event** property containing the specification of the event that triggers the action.



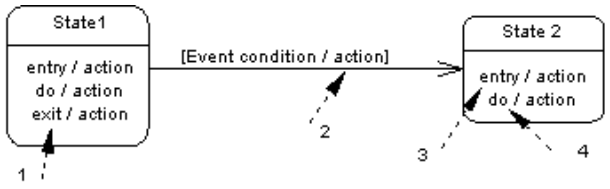
Action on state and on transition

In an OOM, an action is used in the statechart diagram in association with states: the action is executed in the state during entry or exit. It is also used in association with transitions: the action is executed when the transition is triggered.

In UML, the difference is that an action appears in interaction diagrams (in association with messages) and in statechart diagrams.

When you define an action on a state, you can define several actions without any limitation. When you define an action on a transition, there can only be **one** action as the transition can execute only one action. An action defined on a state can contain the event that triggers it: the action property sheet contains the event property sheet. An action defined on a transition does not contain the event that triggers it: you can only enter the action in a text field.

In the following figure, you can see actions defined on states, and actions defined on transitions together with the order of execution of actions:



The action icon in the Browser is a two-wheel symbol, it is defined within a state node but does not appear within a transition node.



Action properties

An action has the following properties:

Property	Description	Maximum length
Name	Name of the action	254
Code	Reference name of the action	254
Comment	Descriptive comment for the action	—
Stereotype	Extends the semantics of an action derived from existing actions but specific to your needs	254
Trigger event	Read-only dropdown listbox that lists events including entry, do, and exit values. It describes the role an action plays for a state or the event that triggers its execution. You can click the Properties tool beside this box to display the event property sheet or you can click the Select Trigger Event tool to select an event in another package	—
Event arguments	Arguments of an event defined on a state. Arguments are instances of parameters or names given to parameters in the context of executing an event. You can specify a list of event arguments (arg1, arg2,...) in this box	254
Operation	Read-only dropdown listbox that lists operations of the classifier associated with the state. It allows you to specify the action implementation using an operation. It is grayed and empty when the classifier is not a class	254
Operation arguments	Arguments of an event defined on an operation	254

Trigger event property of the action

If you use the Trigger Event dropdown listbox and its corresponding buttons in the property sheet of an action, you can do the following:

- Add an event to an action by choosing the event name from the Trigger Event dropdown listbox
- Create a new event by clicking the Create tool beside the Trigger Event dropdown listbox
- Select an event created in the current model or other models by clicking the Select Trigger Event tool beside the Trigger Event dropdown listbox. This opens a selection window. You can then select an event from all open models in the workspace

When a trigger event is defined on an action, the inverse relationship is displayed in the event property sheet (in the Triggered Objects tabbed page of the Dependencies page).

For more information on the trigger event property, see section Defining events.

Condition page of the action

You can specify an additional condition on the execution of an action when the event specified by the trigger event occurs. The Condition page is available for actions defined on states. It displays the following property:

Alias

The Alias allows you to edit or summarize a condition attached to an action. You can also use a text editor in which you detail the condition. For example, you can write information on the condition to execute, as well as open, insert and save any text files containing valuable information.

It is recommended that you write an alias (short expression) when you use a long condition so as to display the alias instead of the condition in the diagram.

The condition of an action is displayed between brackets:



Creating an action on a state

You create an action on a state from the state property sheet. To display the action property sheet, you can:

- Double-click the state symbol
- Right-click the state node in the Browser and select Properties

At creation, an action has a default name including a number, this number is assigned in the order of creation.

❖ **To create an action on a state:**

1 Click the Actions tab in the state property sheet to display the Actions page.

2 Click a blank line in the list.

An arrow appears at the beginning of the line.

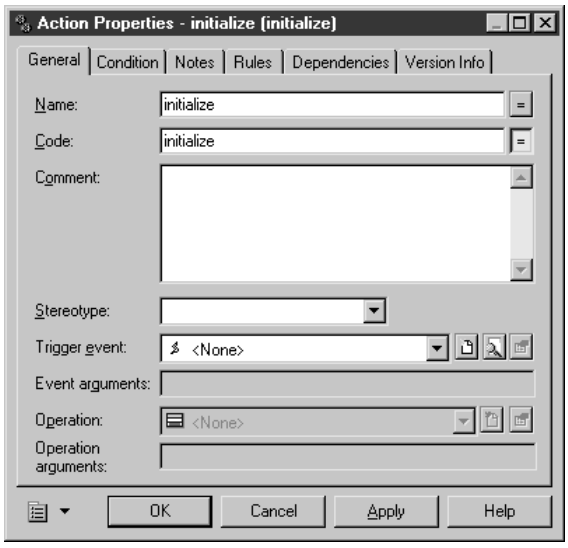
3 Double-click the arrow at the beginning of the line.

A confirmation box asks you to commit the object creation.

4 Click Yes.

The action property sheet appears.

5 Type a name and a code.



6 Click OK.

The action is added to the state.

Accessing the Actions page
You can access the Actions page of a state by right clicking the state symbol in the diagram, and selecting Actions from the contextual menu.

Defining an action on a transition

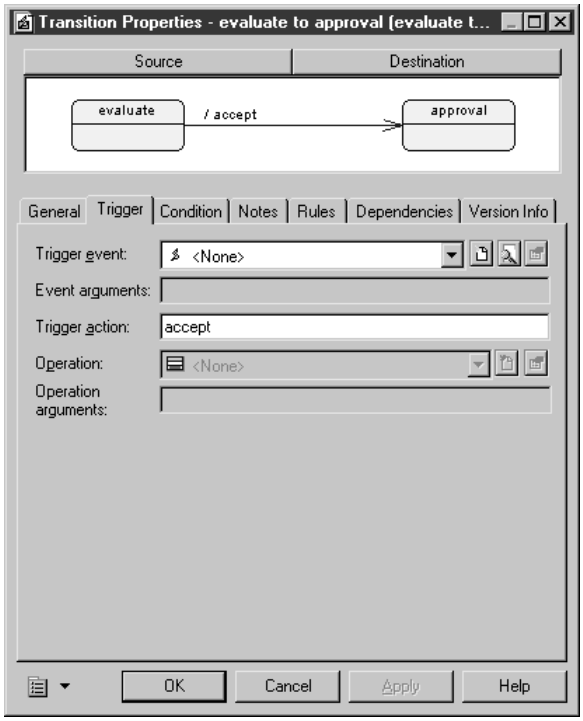
You define an action on a transition from the transition property sheet.

❖ **To define an action on a transition:**

1 Double-click the transition symbol to display the transition property sheet.

2 Click the Trigger tab to display the Trigger page.

3 Type an action name in the Trigger Action box.



4 Click OK.

The action is added to the transition.

Modifying action properties

You can modify the action properties using the following methods:

- From the action property sheet
- From the list of actions

For more information on the different ways to modify the action properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

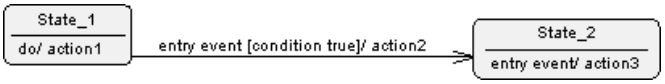
Defining transitions in a statechart diagram

A **transition** is an oriented link between states indicating that an element in a state can enter another state when an event occurs (and if a guard condition is satisfied, when there is one). The expression commonly used in this case is that a transition is fired.

The activity diagram is a simplification of the statechart diagram in which the states have only one action and the transition has a triggered event corresponding to the end of the action. The statechart diagram transition is quite similar to the activity diagram one with the addition of a few properties:

- A trigger event: it is the event that triggers the transition
- A trigger action: it specifies the action to execute when the transition is triggered

The transition link is represented as a simple line with a direction: an arrow. It contains information displayed above its symbol: the associated event, the condition and the action to execute.



You can draw a transition from and to the following objects:

From\to	Start	State	Junction point	End
Start	—	✓	✓	—
State	—	✓	✓	✓
Junction point	—	✓	✓	✓
End	—	—	—	—

✓ = allowed

— = not allowed

Dragging a transition to a different object
You can change the object at either end of a transition by clicking the transition to select it, holding the CTRL key down, and dragging one of the attach points to a different object.

Transitions behavior

The following rules apply to transitions:

- **Reflexive transitions** only exist on states (same source and destination state)
- A trigger event can only be defined if the source is a start or a state
- When you copy a transition, the trigger event is also copied

Merging models
When transitions are compared and merged by the Merge Model feature, they are matched by trigger event first, and then by their calculated name. When two transitions match, the trigger actions automatically match because there cannot be more than one trigger action.

Transition properties in a statechart diagram

You can double-click any transition symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the transition. Calculated, read-only value	254
Code	Reference name of the transition. Calculated, read-only value	254
Comment	Descriptive comment for the transition	—
Source	Where the transition starts from. You can click the Properties tool beside this box to display the object property sheet	254
Destination	Where the transition ends on. You can click the Properties tool beside this box to display the object property sheet	254
Stereotype	Sub-classification derived from an existing transition. It lets you extend the semantics of an object without changing its structure and can be predefined or user-defined	254
Flow type	Defines the type of flow for the transition	—
Trigger event	Read-only dropdown listbox listing all events in the package. It specifies the event that triggers the transition. You can click the Properties tool beside this box to display the event property sheet	254

Property	Description	Maximum length
Event arguments	Arguments of an event defined on a transition. You can specify a list of event arguments (arg1, arg2,...) in this box	254
Trigger action	Text field. Specifies the action to execute when the transition is triggered	254
Operation	Read-only dropdown listbox that lists operations of the classifier associated with the state that is the source of the transition. It allows you to specify the action implementation using an operation. It is grayed and empty when the classifier is not a class	254
Operation arguments	Arguments of an event defined on an operation	254

Flow type property of the transition

You can create your own type of flow in the Flow Type dropdown listbox, or you can choose one of the following default values:

Flow type	Description
Success	Defines a successful flow
Timeout	Defines a timeout limit
Exception	Represents an exception case

Condition page of the transition

A condition is a predicate that applies to a transition. When there are several transitions, the flow evaluates each condition in order to choose the one it will transit on.

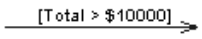
The same concept of condition applies to transitions in statechart and activity diagrams.

Alias

The Alias allows you to edit a condition attached to a transition. It summarizes the condition attached to a transition. You can also use a text editor in which you detail the condition. For example, you can write information on the condition to execute, as well as open, insert and save any text files containing valuable information.

It is recommended that you write an alias (short expression) when you use a long condition so as to display the alias instead of the condition in the diagram.

The condition is displayed near the source symbol as shown below:



Opening the Condition page
You can open the Condition page by right clicking the transition symbol in the diagram, and selecting Condition from the contextual menu.

Trigger page of the transition

The Trigger page in the transition property sheet comprises the following properties:

Property	Description
Trigger event	Read-only dropdown listbox listing all events in the package. It specifies the event that triggers the transition. You can click the Properties tool beside this box to display the event property sheet. It is available only for transitions coming from a state or a start and is not editable in other cases. When you define a trigger event, the inverse relationship is displayed in the Triggered Objects page of the corresponding event property sheet. The Triggered Objects page lists transitions that the event can trigger
Event arguments	Arguments of an event defined on a transition. You can specify a list of event arguments (arg1, arg2,...) in this box
Trigger action	Specifies the action to execute when the transition is triggered
Operation	Read-only dropdown listbox that lists operations of the classifier associated with the state that is the source of the transition. It allows you to specify the action implementation using an operation. It is grayed and empty when the classifier is not a class
Operation arguments	Arguments of an event defined on an operation

For more information on events, see section Defining events.

For more information on actions, see section Defining actions.

Creating a transition in a statechart diagram

You can create a transition:

- From a diagram
- From the Browser

Transitions name and code are calculated. The name and code boxes display a read-only value named after both extremities of the transition.

For more information on the different ways to create a transition, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a transition from a statechart diagram:**

1 Click the Transition tool.

2 Click inside the first state and while continuing to hold down the mouse button, drag the cursor to the second state. Release the mouse button inside the second state.

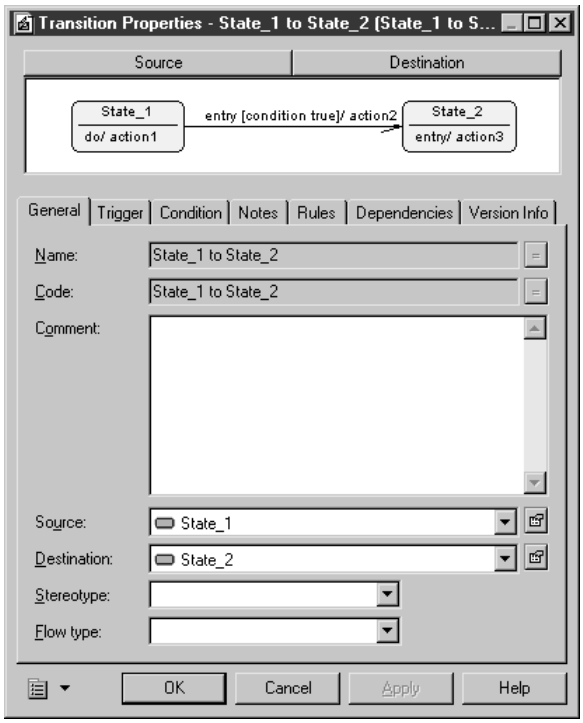
The transition link appears between both objects.

> **Dragging a transition to a different state**
> You can change the state at either end of a transition by clicking the transition to select it, holding the CTRL key down, and dragging one of the attach points to a different state.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Transition tool.

4 Double-click the transition in the diagram to display its property sheet.



You can open the property sheet of the source and destination objects by clicking the Source and Destination buttons located in the upper part of the transition property sheet.

5 Type or select transition properties as required in the different tabbed pages.

6 Click OK.

The transition is also displayed in the property sheet of the state, you can display it from the Input Transitions or Output Transitions tabbed page of the Dependencies page.

Modifying transition properties in a statechart diagram

You can modify the transition properties using the following methods:

- From the transition property sheet
- From the list of transitions

For more information on the different ways to modify the transition properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

If your model contains transitions from statechart and activity diagrams, you can filter the transition you need from the List of Transitions. Depending on where you select the command, the content of the list of transitions varies as follows:

Select the command	Displays
Right-click the model in the Browser and select List of→Transitions	All transitions from statechart and activity diagrams
Double-click a statechart diagram in the Browser and select Model→Transitions	A typed list of transitions from the statechart diagram. The title of the list indicates the diagram type

Modifying transition display preferences in a statechart diagram

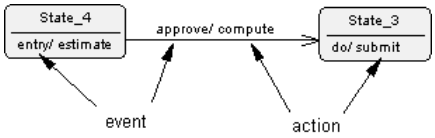
You can modify the following display preference for transitions using the Tools→Display Preferences command:

Preference	Description
Show stereotype	Displays the stereotype of the transition
Show event arguments	Displays the arguments of the event (defined on transitions)

For more information on event arguments, see section Defining event arguments.

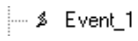
Defining events

An **event** is the occurrence of something observable. The occurrence is assumed to be instantaneous and should not have duration. Events convey information specified by parameters. They are used in the statechart diagram in association with transitions: they are attached to transitions to specify which event fires the transition. They are also used in association with actions: the event can trigger the change of state of a classifier or the execution of an internal action on a state.



The same event can be shared between several transitions and actions. It is reusable by nature because it is not dependent on the context.

The event icon in the Browser is the following symbol:



Predefined events

You can select an event from the Trigger Event dropdown listbox in the action and transition property sheets. You can also select a predefined event value from the Trigger Event dropdown listbox if you define the event on an action.

The list of events contains the following predefined values:

- Entry: the action is executed when the state is entered
- Do: a set of actions is executed after the entry action
- Exit: the action is executed when the state is exited

Examples

Here are some examples of what an event can be:

- A boolean expression becoming true
- The reception of a signal
- The invocation of an operation
- A time event, like a timeout or a date reached

You can display the arguments of an event in the statechart diagram.

For more information on arguments of an event, see section Defining event arguments.

Event properties

You access event properties from an action or a transition property sheet.

Property	Description	Maximum length
Name	Name of the event	254
Code	Reference name of the event	254
Comment	Descriptive comment for the event	—
Stereotype	Sub-classification derived from an existing event. It lets you extend the semantics of an object without changing its structure and can be predefined or user-defined	254

The Dependencies page in the event property sheet contains a Triggered Objects tabbed page that displays the actions on states and on transitions that are triggered by this event.

Defining event parameters

The Parameters page in the event property sheet allows you to define event parameters that correspond to the event signature.

Event parameter properties

A parameter has the following properties:

Property	Description	Maximum length
Parent	Event to which the parameter belongs	254
Name	Name of the parameter	254
Code	Reference name of the parameter	254
Comment	Descriptive comment for the parameter	—
Data type	Set of instances sharing the same semantics	—
Array	When selected, indicates that the data type is a table format	—

Property	Description	Maximum length
Parameter Type	Direction of information flow for the parameter	—
Default value	Default value when a parameter is omitted	—
WSDL data type	Only available with Web services. XML Schema/SOAP type used during invocation of the web method (using http or Soap)	—

Default value

Here is an example to describe the Default value property:

Use an operation `oper(string param1, integer param2)`, and specify two arguments `oper(val1, val2)` during invocation. Some languages, like C++, allow you to define a default value that is then memorized when the parameter is omitted during invocation.

If the declaration of the method is `oper(string param1, integer param2 = default)`, then the invocation `oper(val1)` is similar to `oper(val1, default)`.

Parameter type

The value in the Parameter type dropdown listbox indicates what is returned when the parameter is called by the event during the execution process.

You can set the following values for the direction:

Value	Description
In	Input parameter passed by value. The final value can not be modified and information is not available to the caller
In/Out	Input parameter that may be modified. The final value may be modified to communicate information to the caller
Out	Output parameter. The final value may be modified to communicate information to the caller

Creating an event parameter

You create a parameter from the event property sheet. To display the event property sheet, you can:

- Click the Properties button beside the Trigger Event dropdown listbox in the action or transition property sheet
- Right-click the event node in the Browser and select Properties

❖ **To create an event parameter:**

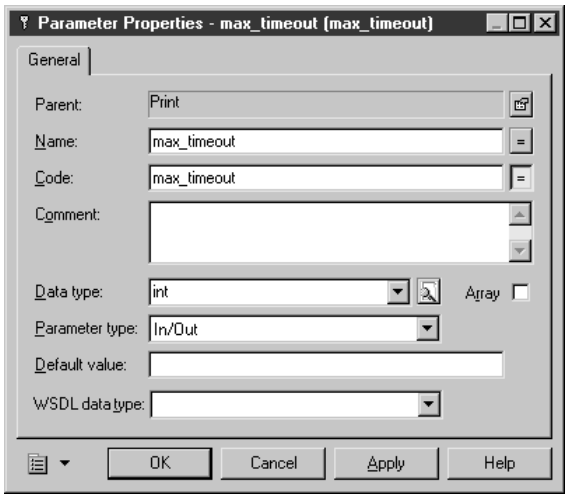
1 Select the Parameters tab in the event property sheet.

2 Click the Add a Row tool.
or
Select a row.

An arrow appears at the beginning of the line.

3 Double-click the arrow at the beginning of the line to display the parameter property sheet.

4 Type a name and code.



5 <optional> Select a value from the Data type and Parameter type dropdown listboxes.

6 Click OK.

The parameter is added to the event.

Modifying event parameter properties

You can modify event parameter properties from the parameter property sheet.

❖ **To modify an event parameter from the property sheet:**

1 Select the Parameters tab in the event property sheet.

2 Select the parameter you want to modify and click the Properties tool to display the parameter property sheet.

3 Modify parameter properties as required.

4 Click OK in each of the dialog boxes.

Defining event arguments

Event arguments are slightly different from event parameters. Event arguments are defined on the action or on the transition that receives the event, they are dependent on the particular context that follows this receipt.

It is a text field defined on the action or the transition. You can edit it and separate arguments with a comma, for example: arg1, arg2. There is no control of coherence between event parameters and event arguments in PowerDesigner.

Example

An event can have a parameter 'person' that is for example, a person sending a request. Within the context of a transition triggered by this event, you may clearly know that this parameter is a customer, and then purposefully call it 'customer' instead of 'person'.

Creating an event

You create an event from the action or the transition property sheet. To display the action or transition property sheet, you can:

- Double-click the state symbol, click the Actions tab and double-click the arrow at the beginning of the line
- Right-click the transition node in the Browser and select Properties

❖ **To create an event:**

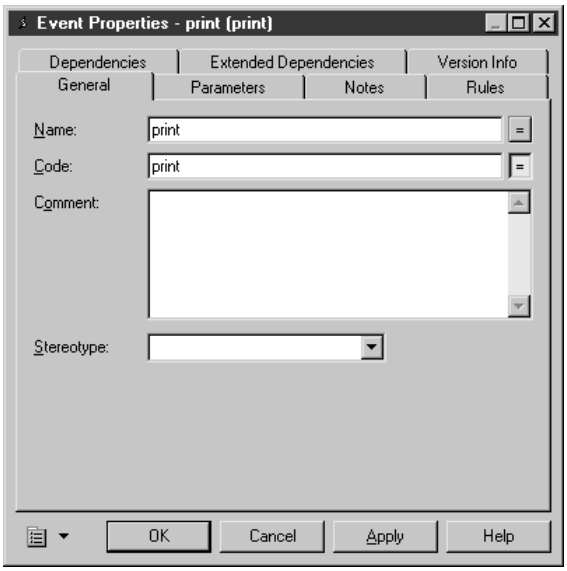
1 Select the Create tool beside the Trigger Event dropdown listbox in the General page of the action property sheet, or in the Trigger page of the transition property sheet.

A confirmation box asks you to commit the object creation.

2 Click Yes.

The event property sheet appears.

3 Type a name and code.



4 Click OK in each of the dialog boxes.

Modifying event properties

You can modify the event properties using the following methods:

- From the event property sheet
- From the list of events

For more information on the different ways to modify the event properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

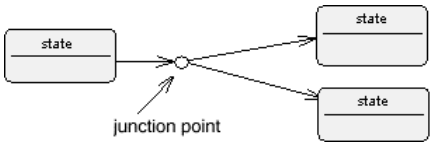
Defining junction points

A **junction point** is similar to the decision in the activity diagram, except that it accepts several input and output transitions. Junction points are used to merge and split several transition paths in a statechart diagram.

You are not allowed to use shortcuts of a junction point. A junction point may be dependent on event parameters if the parameters include some split or merge variables for example.

You can attach two transitions of opposite directions to the same junction point symbol.

The symbol of a junction point is an empty circle:



Junction point properties

You can double-click any junction point symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the junction point	254
Code	Reference name of the junction point	254
Comment	Descriptive comment for the junction point	—
Stereotype	Extends the semantics of a junction point derived from existing junction points but specific to your needs	254

Creating a junction point

There are several ways to create a junction point:

- From a diagram
- From the list of junction points

At creation, a junction point has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a junction point, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a junction point:**

1 Click the Junction Point tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:

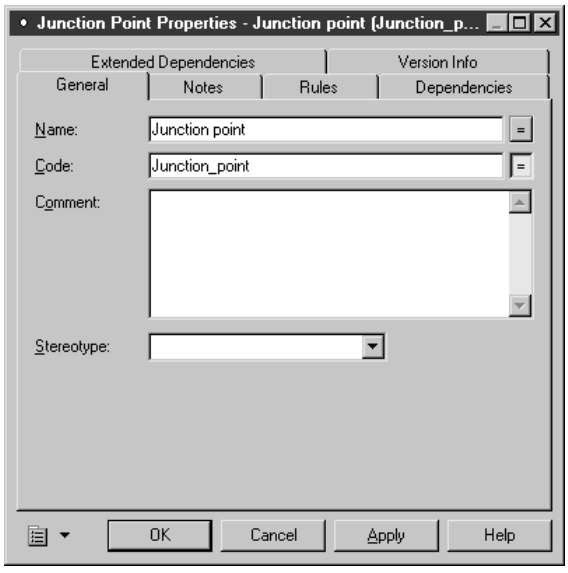


3 Click the Pointer tool.
or
Click the right mouse button.

You release the Junction Point tool.

4 Double-click the junction point symbol to display its property sheet.

5 Type a name and a code.



6 Click OK.

Modifying junction point properties

You can modify the junction point properties using the following methods:

- From the junction point property sheet
- From the list of junction points

For more information on the different ways to modify the junction point properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying junction point display preferences

You can modify the following display preference for junction points using the Tools→Display Preferences command:

Preference	Description
Show name	Displays the name of the junction point
Show stereotype	Displays the stereotype of the junction point

Defining ends in a statechart diagram

An **end** is a termination point of the states described in the statechart diagram. Its symbol is a solid ball inside a circle (bull's eye).



There can be several ends in the same statechart diagram if you want to show divergent end cases, like errors scenarios. There can also be no end at all if you want to show an endless process.

You are not allowed to use shortcuts of an end.

The end has the same meaning in the statechart diagram and in the activity diagram, that is why you can use the same end simultaneously in a statechart and in an activity diagram.

End properties in a statechart diagram

You can double-click any end symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the end	254
Code	Reference name of the end	254
Comment	Descriptive comment for the end	__
Stereotype	Extends the semantics of an end derived from existing ends but specific to your needs	254

Creating an end in a statechart diagram

There are several ways to create an end:

- From a diagram
- From the Browser
- From the list of ends

At creation, an end has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create an end, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ To create an end from a statechart diagram:

1 Click the End tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:

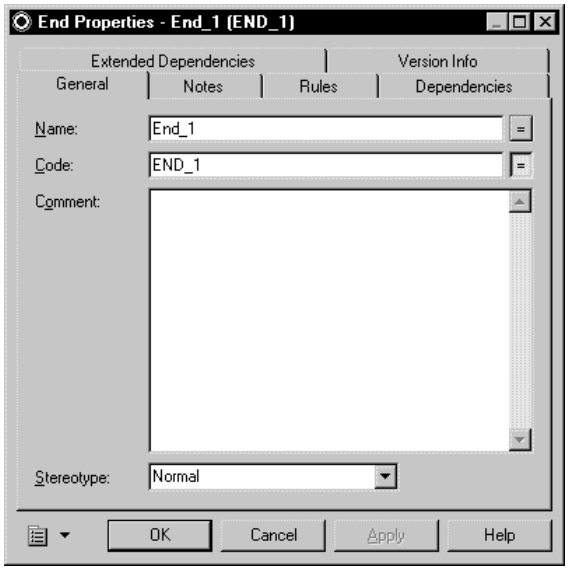


3 Click the Pointer tool.
or
Click the right mouse button.

You release the End tool.

4 Double-click the end symbol to display the end property sheet.

5 Type a name and a code.



6 Click OK.

Modifying end properties in a statechart diagram

You can modify the end properties using the following methods:

- From the end property sheet
- From the list of ends

For more information on the different ways to modify the end properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying end display preferences

You can modify the following display preferences for ends using the Tools→Display Preferences command.

The same display preferences apply to starts and ends in statechart and activity diagrams.

Preference	Description
Show name	Displays the name of the end
Show stereotype	Displays the stereotype of the end

CHAPTER 8

Building an Activity Diagram

About this chapter

This chapter describes how to build an activity diagram. It explains the role of each element in an activity diagram and how to create and modify them.

Contents

Topic	Page
Activity diagram basics	298
Defining starts in an activity diagram	303
Defining activities	306
Defining object states	316
Defining organization units in an activity diagram	321
Defining transitions in an activity diagram	332
Defining decisions in an activity diagram	338
Defining synchronizations in an activity diagram	344
Defining ends in an activity diagram	349

Activity diagram basics

The **activity diagram** is a diagram representing a system behavior, particularly suitable for analysis or documentation purposes. It shows control flows (called transitions) between actions performed in the system (called activities) from a start point to several potential end points.

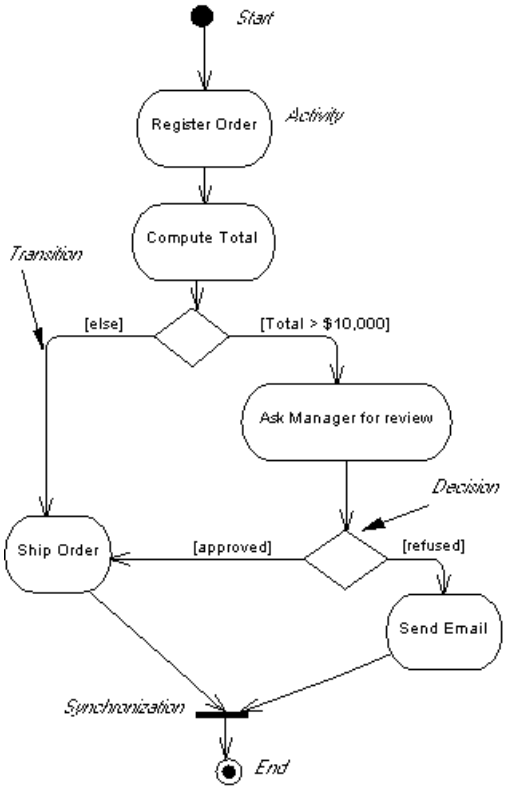
Defining an activity diagram

The activity diagram is one of the diagrams of the Unified Modeling Language (UML). It is a simplification of the UML statechart diagram for modeling control flows in computational and organizational processes.

The activity diagram is part of an Object-Oriented Model. It is one of the available diagrams in an OOM. It shows the symbols of the objects defined in the model. The main objects are the activities and the transitions between them.

At a conceptual level, an activity diagram shows a detailed view of the model. It describes the behavior of a use case, a package, a classifier, or an implementation of an operation. It represents the activities as performing operations (or methods) in which the transitions are triggered by the completion of the operations.

An activity diagram focuses on the operations that are passed among activities.



Why build an activity diagram?

An activity diagram is used to describe processes in which activities represent the completion of internally generated actions. It is meant to represent the internal behavior of a method (implementation of an operation), a classifier or a use case.

It is a variation of the statechart diagram that focuses on the implementation of operations in which most of the events correspond precisely to the end of the preceding activity, whereas the activity diagram does not differentiate the states, the activities and the events. The activity diagram is built to show a simplified representation of a process, using flows from activity to activity.

For more information on diagrams of the OOM, see section What is an OOM, in chapter Object-Oriented Model Basics.

Activity diagrams vs sub-activity diagrams

In an activity diagram, you can have an activity diagram created within a model or a package, and a sub-activity diagram created within a composite activity.

Activity diagrams

You can create several activity diagrams in a package or a model. Each of those diagrams is independent and designs an activity. This activity defines an isolated context in which the integrity of elements can be checked.

In an activity diagram, it is possible to:

- Create packages
- Convert the activity diagram to a package

  For more information on converting to a package, see section Converting a diagram to a package, in chapter Managing Models in the *General Features Guide*.
- Convert the activity diagram to a composite activity

  For more information on converting to a composite activity, see section Converting an activity diagram to a composite activity.

Sub-activity diagrams

A sub-activity diagram designs an activity decomposition. It details the implementation of a parent activity, called composite activity. All integrity checks are performed on all elements defined in the composite activity.

In a sub-activity diagram, it is **not** possible to create packages. That is why the package tool appears gray in the tool palette once you are in a sub-activity diagram. You can only use shortcuts of packages.

You could create several sub- activity diagrams within a composite activity, however it is generally of no interest to define several sub-activity diagrams under the same parent activity.

For more information on sub-activity diagrams, see section Defining composite activities.

Creating an activity diagram

You can create an activity diagram using different methods.

For more information on creating an activity diagram in a new OOM, see section Creating an OOM, in chapter Object-Oriented Model Basics.

For more information on creating an activity diagram in an existing OOM, see section Creating a new diagram, in chapter Managing Models, in the *General Features Guide*.

Merging activity diagrams
When you use the Merge Models feature to merge activity diagrams, the symbols of all elements in the activity diagram are merged without comparison. You can either accept all modifications on all symbols or no modification at all.

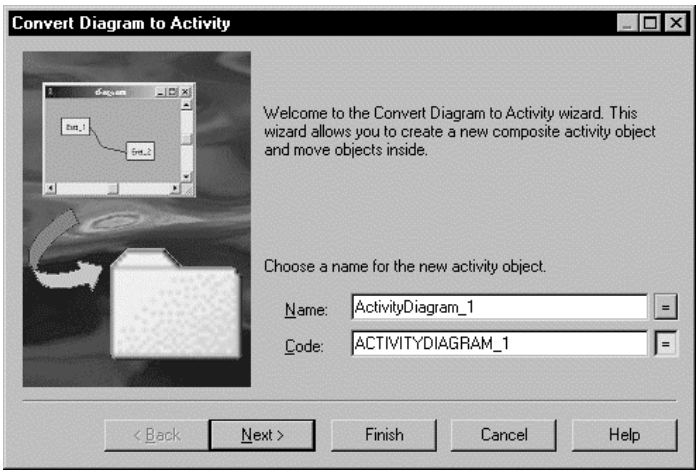
Converting an activity diagram to a composite activity

PowerDesigner lets you convert a diagram to a composite activity using the Convert to Composite Activity Wizard from the Tools menu. You do so, when you want to describe the context of a complex activity. The diagram must contain objects to proceed to the conversion in the wizard.

❖ **To convert an activity diagram to a composite activity:**

1 Right-click the diagram node in the Browser and select Convert to Composite Activity from the contextual menu.
or
Right-click the diagram background window and select Diagram→Convert to Composite Activity from the contextual menu.
or
Select Tools→Convert to Composite Activity.

The Convert Diagram to Activity page appears.


Convert Diagram to Activity
Welcome to the Convert Diagram to Activity wizard. This wizard allows you to create a new composite activity object and move objects inside.
Choose a name for the new activity object.
Name:
ActivityDiagram_1
Code:
ACTIVITYDIAGRAM_1
< Back
Next >
Finish
Cancel
Help

2 Type a name and a code.

3 Click Next if you have to select objects of the diagram to be moved into the composite activity.
or
Click Finish.

The selected objects are moved into the composite activity.

Defining starts in an activity diagram

A **start** is a starting point of the whole process represented in the activity diagram. Its symbol is a solid ball as shown below:



How many starts are allowed?

You can create several starts within a package or a model since you may use several activity diagrams in the same model or package, but only **one** start is allowed in the activity diagram. As per the composite activity, you can only create **one** start in the default sub-activity diagram.

For this reason, the Start tool is grayed in the palette if a start symbol already exists in the current diagram. Pasting a start is also forbidden if a start already exists in the activity or the sub-activity diagram.

You are not allowed to use shortcuts of a start.

Merging models
The start is compared and merged in the Merge Model feature. Since starts should be unique in composite activities, the Merge Model feature checks that there is no additional start in a composite activity.

The start shares the same concept in the statechart diagram and in the activity diagram, that is why you can use the same start simultaneously in a statechart and in an activity diagram.

For more information on starts in the statechart diagram, see section Defining starts in a statechart diagram, in chapter Building a statechart diagram.

Start properties in an activity diagram

You can double-click any start symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the start	254
Code	Reference name of the start	254
Comment	Descriptive comment for the start	—
Stereotype	Extends the semantics of a start derived from existing starts but specific to your needs	254

Creating a start in an activity diagram

You create a start from a diagram using the tool palette.

❖ **To create a start from an activity diagram:**

1 Click the Start tool.

2 Click anywhere in the diagram.

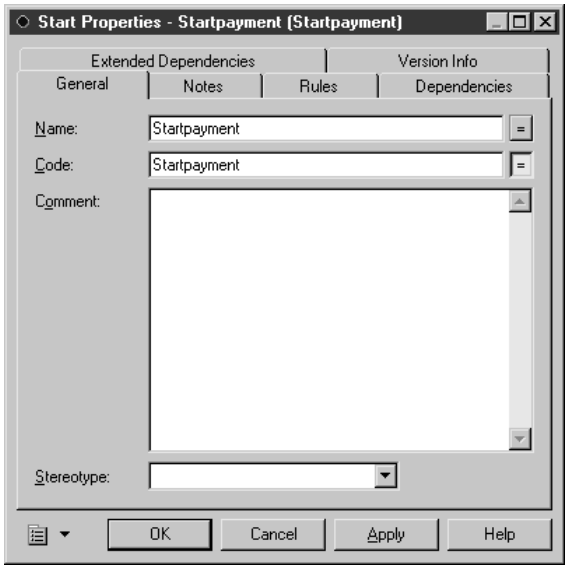
The following symbol appears at the click position:



The Start tool is grayed in the palette after creation since you can only create one start per diagram.

3 Double-click the start symbol to display the start property sheet.

4 Type a name and a code.



5 Click OK.

Modifying start properties in an activity diagram

You can modify the start properties using the following methods:

- From the start property sheet
- From the list of starts

For more information on the different ways to modify the start properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying start display preferences

You can modify the following display preferences of a start using the Tools→Display Preferences command.

The same display preferences apply to starts and ends in statechart and activity diagrams.

Preference	Description
Show name	Displays the name of the start
Show stereotype	Displays the stereotype of the start

Defining activities

An **activity** is the invocation of a manual or automated action. When the activity gains the control, it performs the action, then, depending on the result of the action, the transition (control flow) is passed to another activity.

An activity can be atomic or composite:

- An atomic activity does not contain sub-activities
- A composite activity uses sub-activities to describe its actions

For more information on composite activities, see section Defining composite activities.

Example

An activity may be the invocation of an action like "send a mail", or "increment a counter".

The symbol of an activity is the following:



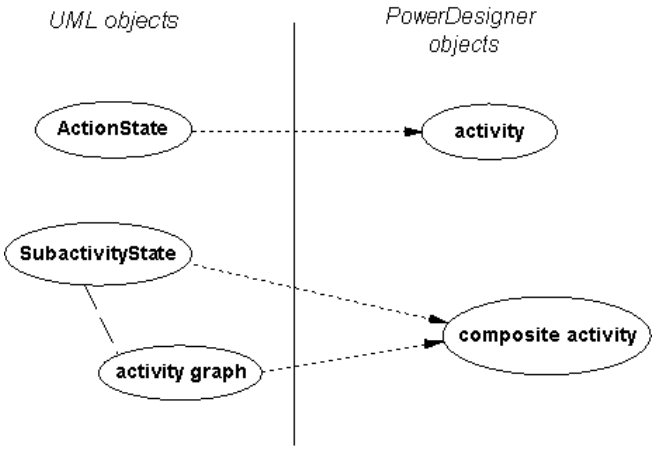
You can use internal and external shortcuts of activities in your model.

Using external shortcuts of activities
You can use shortcuts of activities that belong to an external model and include them in the current model. For more information on shortcuts, see section Managing shortcuts in the *General Features Guide*.

About the UML notation

An activity in an OOM is the equivalent of the UML activity (ActionState or SubactivityState) and the activity graph. In UML, the ActionState represents the execution of an atomic action, the SubactivityState is the execution of an activity graph and the activity graph is the description of a complex action (designed by sub-activities).

The following chart highlights UML vs PowerDesigner terminology and concepts:



In PowerDesigner, the merge of SubactivityState and activity graph into a composite activity allows you to define sub-activities directly under the parent activity without defining an additional object. If you do need to highlight the difference, you can create activities directly under the model or the package, and use activity shortcuts to detail the activity implementation. In this case, the SubactivityState corresponds to the shortcut of a composite activity.

Activity properties

You can double-click any activity symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the activity	254
Code	Reference name of the activity	254
Comment	Descriptive comment for the activity	__
Stereotype	Sub-classification derived from an existing activity. It lets you extend the semantics of an object without changing its structure and can be predefined or user-defined	254

Property	Description	Maximum length
Organization unit	Displays the organization units defined in the model. You can click the Properties tool beside this box to display the organization unit property sheet	254
Composite	If selected, implies that the activity becomes a composite activity	—

Modify or create new stereotypes

You can modify an existing stereotype or create a new one from the object language. You can also use the extended model definition to create new sets of stereotypes.

For more information on how to manipulate categories and entries in the object language, see chapter The Resource Editor in the *General Features Guide*.

For more information on extended model definitions, see section Working with extended model definitions, in chapter Managing Objects in the *General Features Guide*.

Organization unit property of the activity

The Organization unit dropdown listbox lists the organization units defined in the model. It is used to specify which organization unit is linked to an activity. You can click the Ellipsis button beside the Organization unit dropdown listbox to create a new organization unit in the List of Organization Units.

For more information on organization units, see section Defining organization units in an activity diagram.

Action page of the activity

The action page defines the nature, the type and the duration of an action that an activity executes.

It contains the following values:

- Action type: specifies the way the action should be executed. It can be a manual, automated or an undefined type. The Action type zone also contains a text editor in which you detail how the action is executed. For example, you can write a pseudo code or an information on the program to execute, as well as open, insert and save any text files.

- Duration: specifies estimated or statistic duration to execute the action. This information is for documentation purposes only; estimate on the global duration is not computed
- Timeout: set to zero by default. If the value is not set to zero, it means that a timeout exception occurs if the execution of the activation takes more than the specified timeout limit. You can type any alphanumeric value in the Timeout box (example: 20 seconds).

Opening the Action page
You can open the Action page by right clicking the activity symbol in the diagram, and selecting Action from the contextual menu.

Defining composite activities

A composite activity is a child activity used to describe the actions of the activity. It is equivalent to describing a SubactivityState and an activity graph in UML.

For more information on the UML notation, see section About the UML notation.

There is no limitation on the number of decompositions. Each activity can be decomposed into a composite activity, except the lowest level activities. Those ones do not need to be further decomposed because they intrinsically contain enough detail.

The Composite check box in the activity property sheet allows you to create or delete a composite activity as follows:

- If you select the Composite box, a sub-activity diagram is created within the activity to specify an activity decomposition. You can then create any objects of the activity diagram
- If you deselect the Composite box, or select Change to atomic from the contextual menu of a composite activity, a confirmation dialog box asks you to confirm the objects deletion. The existing sub-activity diagram and all its objects are then deleted. There is no confirmation dialog box if no objects have been defined in the sub-activity diagram

The composite activity symbol appears with an icon in the right-hand corner.



A composite activity always contains a default sub-activity diagram. It would be possible to create several sub-activity diagrams within a composite activity, but it is not recommended because being in a sub-activity diagram means being within the context of an activity. Unless you want to design some exception cases like error management for example, it would not be consistent to create too many sub-activity diagrams within a composite activity.

The composite activity behaves like a **specialized** package or container that contains a sub-activity diagram, for this reason, it is not possible to create a package, or any other UML diagram type in a composite activity.

Dragging activity diagram objects into a composite activity
You can drag any activity diagram objects from the Browser and drop them into a sub-activity diagram. These objects are automatically inserted into the default sub-activity diagram.

All sub-activities defined within a composite activity share the same namespace as the parent activity.

Including composite activities
To display all activities in composite activities of the model, click the Include composite activities tool in the list of activities.

When you work within a composite activity, the associated objects appear under the activity node in the Browser. The activity node can be expanded to display its child nodes, this allows you to see the activity decomposition in a quick and convenient way.

Find in Browser
You can locate any object or any diagram in the Browser tree view from the current diagram window. To do so, right-click the object symbol, or the diagram background and select Edit→Find in Browser.

Creating an activity

There are several ways to create an activity:

- From a diagram
- From the Browser
- From the list of activities

At creation, an activity has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create an activity, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an activity from a diagram:**

1 Click the Activity tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



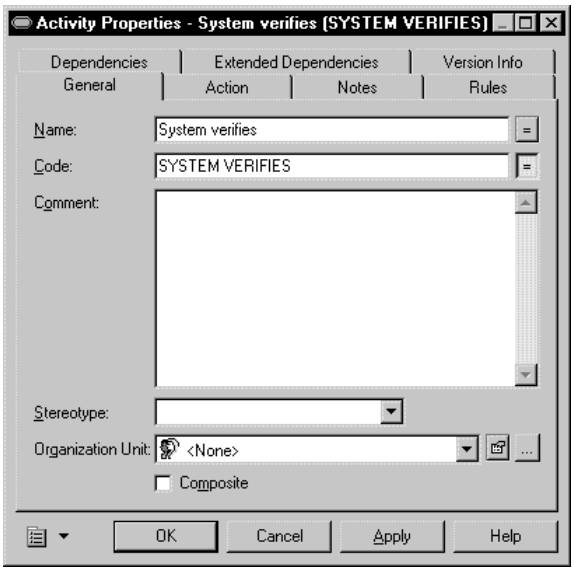
3 Click the Pointer tool.
or
Click the right mouse button.

You release the Activity tool.

4 Double-click the activity symbol to display the activity property sheet.

Opening property sheets at last accessed page
Property sheets open to the General page by default. However, you can choose to open property sheets at the last page accessed by selecting Tools→General Options→Dialog, and selecting the option Keep Last Tab in the Property Sheets groupbox.

5 Type a name and a code.



6 Click OK.

Group Symbols not available
You can not use the Symbol→Group Symbols feature to group activities in the activity diagram.

Creating a composite activity

You create a composite activity from the symbol of the parent activity:

- From the diagram
- From the contextual menu

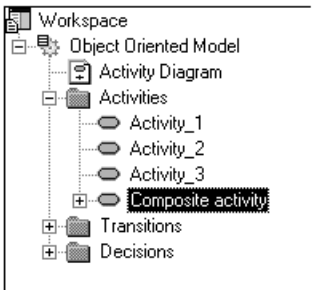
When you create a composite activity, a default sub-activity diagram is automatically created.

Creating a composite activity from the diagram

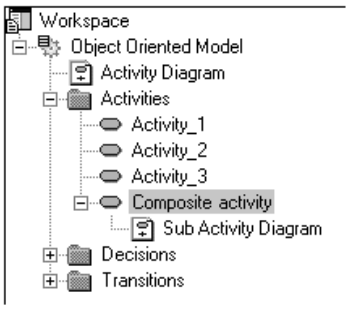
❖ **To create a composite activity from the diagram:**

- Press CTRL and double-click the parent activity symbol to create the composite activity

A default sub-activity diagram opens in the diagram window. The composite activity node is visible in the Activities category of the Browser marked with a + sign.



If you expand the composite activity node, you see that a default sub-activity diagram node is automatically created:



Creating a composite activity from the contextual menu

You can create a composite activity from the activity contextual menu, it is equivalent to selecting the Composite box in the activity property sheet.

❖ **To create a composite activity from the contextual menu:**

- Right-click the parent activity symbol and select Change to Composite from the contextual menu.

  The activity symbol changes to the following symbol:



Opening a composite activity

You can open a composite activity using the following methods:

- Press CTRL and double-click on the composite activity symbol.
  or
  Select Open Diagram from the contextual menu of the composite activity.

  The default sub-activity diagram opens in the diagram window.

Closing a composite activity

You can close a composite activity using the following methods:

- Press CTRL + U in the sub-activity diagram to go up to the default diagram.
  or
  Select Diagram→Go Up One Level from the contextual menu.
  If several diagrams are open in the workspace, you are requested to select the parent activity diagram.

> **Setting a diagram as default diagram**
> You can set a diagram to be the default diagram by selecting the Default Diagram check box in the model property sheet.

Modifying activity properties

You can modify the activity properties using the following methods:

- From the activity property sheet
- From the list of activities

 For more information on the different ways to modify the activity properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying activity display preferences

You can modify the following display preference for activities using the Tools→Display Preferences command:

Preference	Description
Show stereotype	Displays the stereotype of the activity

Defining object states

An **object state** is the association of an object (instance of a class) and a state. It represents an object in a particular state.

Its symbol is a rectangle as shown below:



In the activity diagram, the same object can evolve after several actions defined by activities, have been executed. For example, a document can evolve from the state initial, to draft, to reviewed, and finally turn into a state approved.

If we decompose the term object state, **object** is an instance of a class, and **state** is a specific state of the object.

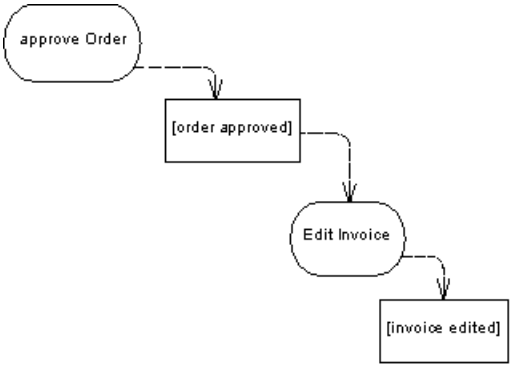
You can draw a link from an activity to an object state and inversely:

A flow from an activity to an object state means that the execution of the activity puts the object in a specific state. It represents the result of an activity.

A flow from an object state to an activity means that the activity uses this specific state in its execution. It represents a data flow between them.

When an activity puts an object in a state and this object is immediately reused by another activity, it shows a transition between two activities with some data exchange, the object state representing the data exchange.

Example

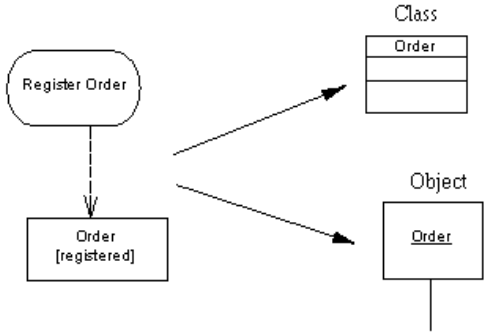


These flows are displayed as dashed lines in the diagram. You use the same transition tool as between two activities, but the line appears dashed after creation.

Global example

How are object states involved in the workflow associated with an activity diagram?

For example, an object state whose state is **[registered]** can be associated with an object **Order**, this object can emerge from the sequence diagram from some activity named **Register Order**. Furthermore this same object can be related to the class Order represented in a class diagram in the model.



Object state properties

You can double-click any object state symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the object state	254
Code	Reference name of the object state	254
Comment	Descriptive comment for the object state	__
Stereotype	Sub-classification derived from an existing object state. It lets you extend the semantics of an object without changing its structure and can be predefined or user-defined	254
Object	Object (instance of a class) related to the object state. You can click the Properties tool beside this box to display the object property sheet	254

Object property of the object state

If you use the Object dropdown listbox and its corresponding buttons in the property sheet of an object state, you can do the following:

- Link an object state to an object (instance of a class) by choosing the object name from the Object dropdown listbox
- Create a new object by clicking the Create tool beside the Object dropdown listbox
- Select any objects created in the current model or other models by clicking the Select Object button beside the Object dropdown listbox. This opens a selection window. You can then select any objects from all open models in the workspace

Syntax of object state names

When you select an object from the property sheet of an object state, the name of the object state shows respectively:

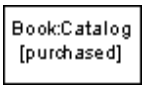
1 The object

2 The class, containing the object

3 The state of the object

Example

An object state linked to an object named Book, associated with a class named Catalog is in a state [purchased]. The name displayed in the symbol of the object state will be:



Drag and drop an object or a class

You can drag an object or a class from the Browser and drop them into an activity diagram. This will automatically create an object state showing the object or the class name.

It may not be possible to move the object directly in the diagram, if this one is a composite activity for example. In this case, you can use the Copy and Paste as Shortcut features to move your object.

For more information on object states, see sections Linking an object to an object state and Linking a class to an object state, in chapter Working with Object-Oriented Models.

Creating an object state

There are several ways to create an object state:

- From a diagram
- From the Browser
- From the list of object states

At creation, an object state has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create an object state, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ To create an object state from a diagram:

1 Click the Object State tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:

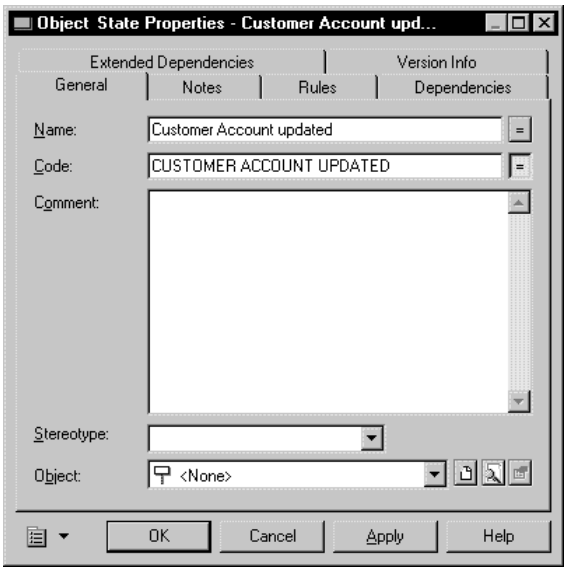


3 Click the Pointer tool.
or
Click the right mouse button.

You release the Object State tool.

4 Double-click the object state symbol to display the property sheet of the object state.

5 Type a name and a code.



6 Click OK.

Modifying object state properties

You can modify the object state properties using the following methods:

- From the property sheet of the object state
- From the list of object states

For more information on the different ways to modify the object state properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying object state display preferences

You can modify the following display preference for object states using the Tools→Display Preferences command:

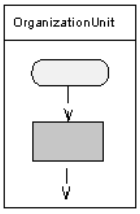
Preference	Description
Show stereotype	Displays the stereotype of the object state

Defining organization units in an activity diagram

An **organization unit** is an optional element that allows you to graphically highlight which organization unit is responsible for which activity. It can represent a company, a system, a service, an organization, a user or a role.

The organization unit is equivalent to the **swimlane** in UML. In the OOM, the organization unit is the tangible element and the swimlane is the symbol that represents the organization unit.

Verify that the Organization Unit Swimlane check box is selected from the Tools→Display Preferences→General command. The graphical symbol of the organization unit will then be a swimlane. It can contain all the symbols of an activity diagram: activities, object states, transitions, etc. as shown below:



You can also display swimlane symbols by right clicking in the activity diagram and select Enable Swimlane Mode from the contextual menu, or conversely by right clicking in the activity diagram and select Disable Swimlane Mode from the contextual menu.

You can create internal and external shortcuts of organization unit.

Using models from version 9
If you use a model that has been created in version 9, use the Tools→Display Preferences→General command and select the Organization Unit Swimlane check box to display the organization units.

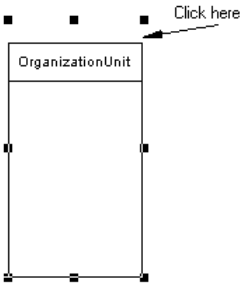
Organization unit properties in an activity diagram

You can open the property sheet of an organization unit using the following method:

- Position the cursor above the organization unit symbol until you see a vertical arrow pointing to the frame.



- Click with the left mouse button to display a selection frame around the symbol.
- Double-click the zone between the selection frame and the swimlane to open the property sheet of the organization unit.



The organization unit property sheet displays the following properties:

Property	Description	Maximum length
Name	Name of the organization unit	254
Code	Reference name of the organization unit	254
Comment	Descriptive comment for the organization unit	—
Stereotype	Extends the semantics of an organization unit derived from existing organization units but specific to your needs	254
Parent organization	Displays the parent organization unit	254

Stereotype

An organization unit has the following default stereotypes:

Stereotype	Description
Role	Defines a role a user plays
User	Defines a user
Group	Defines a group of user
Company	Defines a company
Organization	Defines an organization as a whole
Division	Defines a division in a global structure
Service	Defines a service in a global structure

Parent organization

The Parent Organization dropdown listbox lists all organization units in the model.

Each time an organization unit is selected in this box, it becomes the parent of another organization unit, the latter becoming the child. For example, you may want to describe an organizational hierarchy between a department Dpt1 and a department manager Dptmgr1 with Dptmgr1 as parent organization of Dpt1.

The relationship between parent and child organization units is also displayed in the Dependencies page from the property sheet of the child organization unit.

Creating an organization unit in an activity diagram

When the Organization Unit Swimlane check box is selected in the Display Preferences dialog box, the tool palette displays the Organization Unit tool:



When the Organization Unit Swimlane check box is deselected in the Display Preferences dialog box, the Organization Unit tool is grayed in the tool palette.

You can choose to create an organization unit with or without any swimlane symbol.

Creating an organization unit with a swimlane

❖ **To create an organization unit with a swimlane:**

1 Click the Organization Unit tool in the palette.

2 Click in the diagram.

The following symbol appears at the click position:



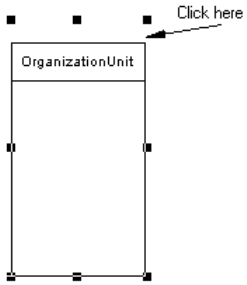
3 Click the right mouse button to release the Organization Unit tool.

4 Position the cursor above the organization unit symbol until you see a vertical arrow pointing to the frame.

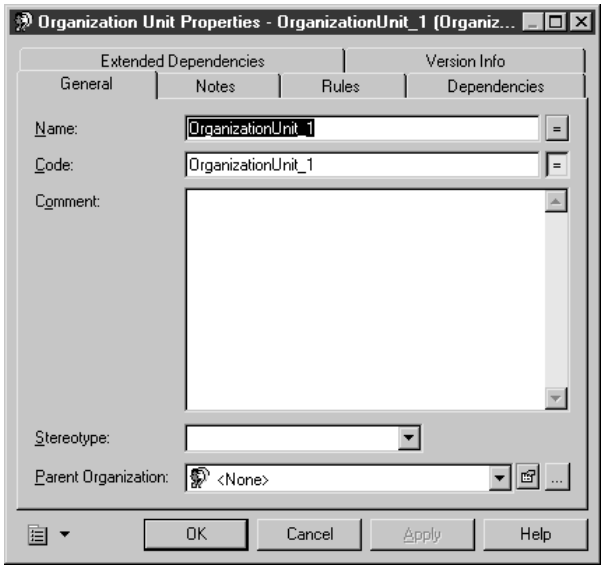


5 Click with the left mouse button to display a selection frame around the symbol.

6 Double-click the zone between the selection frame and the swimlane to open the property sheet of the organization unit.



7 Type a name and a code.



8 <Optional> Type or select a parent organization in the Parent organization dropdown listbox.

9 Click OK.

Group Symbols not available
You cannot use the Symbol→Group Symbols feature to group organization units/swimlanes in the activity diagram.

Creating an organization unit without a swimlane

If you do not need to represent the organization unit in the diagram, you can create an organization unit without a swimlane using the following methods:

- From the Browser
- From the list of organization units

At creation, an organization unit has a default name including a number, this number is assigned in the order of creation.

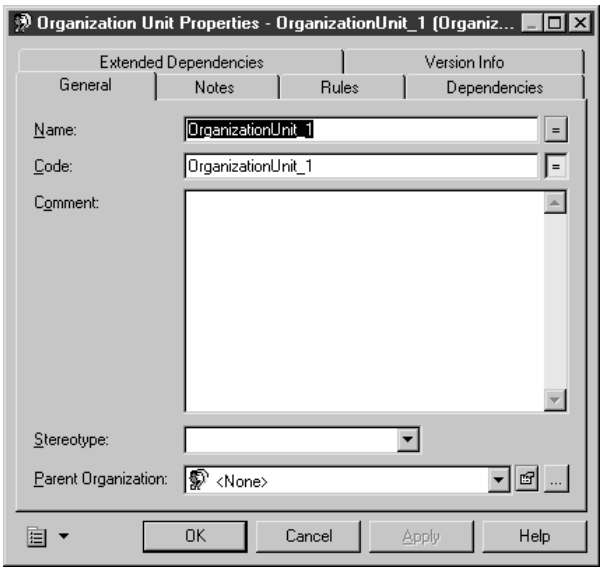
For more information on the different ways to create an organization unit, see section Creating an object in chapter Managing objects in the General Features Guide.

❖ **To create an organization unit without a swimlane:**

1 Right-click the model node in the Browser.

2 Select New→Organization Unit to display the property sheet of the organization unit.

3 Type a name and a code.



4 <Optional> Type or select a parent organization in the Parent organization dropdown listbox.

5 Click OK.

The node of the new organization unit is visible in the Organization Units category of the Browser.

Attaching an activity to an organization unit

When an activity is attached to an organization unit:

- The activity is attached to the swimlane and appears within the swimlane in the activity diagram
- The organization unit name appears in the Organization Unit dropdown listbox in the activity property sheet

❖ **To attach an activity to an organization unit:**

- Double-click the activity in the diagram to open the activity property sheet. Select an organization unit from the Organization Unit dropdown listbox and click OK.
  or
  Click the Activity tool in the palette and click in the required swimlane.

  The organization unit name appears in the Organization Unit dropdown listbox and the activity is attached to the swimlane.

Pressing the ALT key
You can easily attach an existing activity to a swimlane by selecting the activity symbol in the diagram, pressing the ALT key and dragging the activity to the swimlane.

Detaching an activity from an organization unit

❖ **To detach an activity from an organization unit:**

- Press the ALT key then click on the activity symbol in the diagram, and drag the activity outside the swimlane.
  or
  Select <None> in the Organization Unit dropdown listbox in the activity property sheet.

Managing swimlanes in an activity diagram

All symbols displayed in the diagram and associated with an organization unit must be drawn in the corresponding swimlane. Activity symbols are slightly different because you can attach activities to a swimlane, thus if you change the attachment of an activity, its symbol moves to the appropriate swimlane.

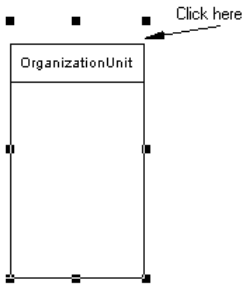
For more information on attaching an activity to a swimlane, see section Attaching an activity to an organization unit.

You can move, copy and paste, resize, delete a swimlane, or change its format preferences.

Selecting swimlanes

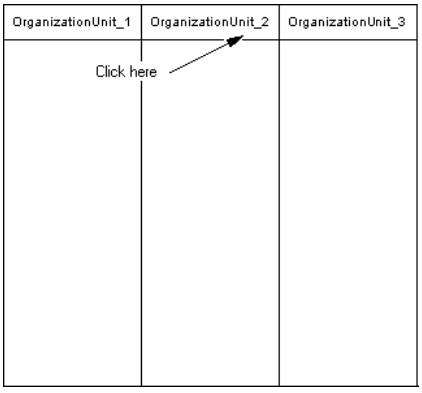
Selecting a single swimlane

To select a swimlane, position the cursor above the organization unit symbol until you see a vertical arrow pointing to the frame, then click the left mouse button.



Selecting all swimlanes

There can only be one whole set of swimlanes per diagram. To select the entire set of swimlanes, click in the header of one of the swimlanes.



Moving, copying and pasting swimlanes

You can move an individual swimlane to another location. You can also move the entire set of swimlanes to another location but it is not possible to move several swimlanes out of the whole set at the same time.

Your can move or copy a swimlane and paste it to the following locations:

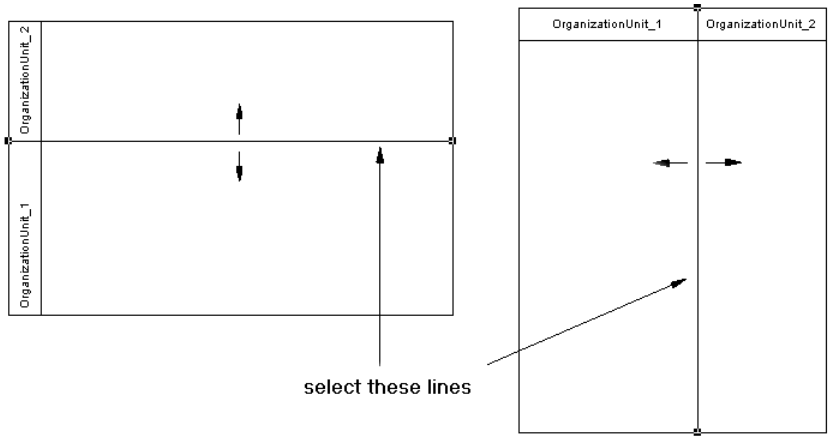
Diagram	What happens...
Different	When you move or copy the swimlane as an individual symbol to another folder or diagram, the symbols that exist inside the swimlane are not copied
Same	When you move a swimlane within the same diagram, you change the order of swimlanes inside the whole set of swimlanes. In this case, all symbols inside the swimlane are moved at the same time (even if some elements are not attached to the swimlane). This is to preserve the layout of the diagram

If you move attached activities inside a swimlane, the width or height of the swimlane varies to include all attached activities.

When you move or copy an entire set of swimlanes and paste it into a different diagram, the symbols that exist inside the swimlanes are not copied.

Resizing swimlanes

You resize a swimlane by selecting one of the two sidelines of the swimlane. Depending if you use vertical or horizontal swimlanes, you can drag the cursor to change the size of the swimlane as shown below:



When you change the width or height of an individual swimlane, all activity symbols attached to the swimlane keep their position.

You can also resize the entire set of swimlanes by selecting one of the handles around them and drag it to any direction.

For more information on the selection of swimlanes, see section Selecting swimlanes.

For more information on the swimlane orientation, see section Modifying organization unit display preferences in an activity diagram.

Changing the format of swimlanes

You can change the format of swimlanes from the Symbol→Format dialog box.

Each swimlane has its own Symbol Format dialog box from which you can choose your preferences.

❖ **To change the symbol format of a swimlane:**

1 Position the cursor above the swimlane until you see a vertical arrow pointing to the frame, click the right mouse button and select Format from the contextual menu to display the Symbol Format dialog box.

2 Type or select changes in the different tabbed pages.

3 Click OK.

Deleting swimlanes

You can delete a swimlane by selecting it and pressing the DELETE key. When you delete a swimlane, the symbols that exist inside are not deleted.

Merging activity diagrams
When you use the Merge Models feature to merge activity diagrams, the symbols of all elements in swimlanes are merged without comparison. You can either accept all modifications on all symbols or no modification at all.

Modifying organization unit properties in an activity diagram

You can modify the organization unit properties using the following methods:

- From the organization unit property sheet
- From the list of organization units

For more information on the different ways to modify the organization unit properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying organization unit display preferences in an activity diagram

You can modify the following display preference for organization units using the Tools→Display Preferences→Organization unit command:

Preference	Description
Show stereotype	Displays the stereotype of the organization unit

You can also define the following display preferences using the Tools→Display Preferences→General command:

Preference	Description
Organization unit swimlane	When selected, displays the organization unit as a swimlane
Horizontal	When the Organization unit swimlane check box is selected, displays swimlanes horizontally in the diagram
Vertical	When the Organization unit swimlane check box is selected, displays swimlanes vertically in the diagram

Defining transitions in an activity diagram

The transition in the activity diagram shares the same concept as in the statechart diagram.

For more information on transitions in the statechart diagram, see section Defining transitions in a statechart diagram, in chapter Building a statechart diagram.

Transition between activities

A **transition** is a link between two activities represented as a line going from one activity to another. It is a route the control flow transits on to link activities. The routing of the control flow is made using guard conditions defined on transitions. If the condition is true, the control is passed to the next element. The transition link is represented as a simple line with a direction (arrow).

Transition between activity and object state

A transition between an activity and an object state indicates that the execution of an activity puts an object in a specific state. When a specific event occurs or when specific conditions are satisfied, the control flow passes from the activity to the object state. Inversely, a transition between an object state and an activity means that the activity uses this specific state in its execution. The transition link is represented as a dashed line with a direction (arrow).

In addition to the above mentioned transitions, you can draw a transition from and to the following objects:

From\to	Start	Activity	OS*	Decision	Fork	Join	End
Start	—	✔	✔	✔	✔	—	—
Activity	—	✔	✔	✔	✔	✔	✔
OS*	—	✔	—	✔	✔	✔	—
Decision	—	✔	✔	✔	✔	✔	✔
Fork	—	✔	✔	✔	✔	✔	—
Join	—	✔	✔	✔	✔	✔	✔
End	—	—	—	—	—	—	—

✔ = allowed

— = not allowed

***OS** = Object State

Dragging a transition to a different object
You can change the object at either end of a transition by clicking the transition to select it, holding the CTRL key down, and dragging one of the attach points to a different object.

A transition can link shortcuts. A transition accepts shortcuts on both extremities to prevent the transition from being automatically moved when an activity is to be moved. In this case, the activity is moved and leaves a shortcut, but contrary to the other links, the transition is not moved. Shortcuts of transitions do not exist, and transitions remain in place in all cases.

Transitions behavior

The following rules apply to transitions:

- Two transitions can not be defined between the same source and destination objects (**parallel transitions**). The Merge Model feature forbids creation of parallel transitions
- **Reflexive transitions** only exist on activities (same source and destination activity)

Transition properties in an activity diagram

You can double-click any transition symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the transition. Calculated, read-only value	254
Code	Reference name of the transition. Calculated, read-only value	254
Comment	Descriptive comment for the transition	—
Source	Where the transition starts from. You can click the Properties tool beside this box to display the object property sheet	254
Destination	Where the transition ends on. You can click the Properties tool beside this box to display the object property sheet	254
Stereotype	Sub-classification derived from an existing transition. It lets you extend the semantics of an object without changing its structure and can be predefined or user-defined	254
Flow type	Defines the type of flow for the transition	—

Flow type property of the transition

You can create your own type of flow in the Flow Type dropdown listbox, or you can choose one of the following default values:

Flow type	Description
Success	Defines a successful flow
Timeout	Defines a timeout limit
Exception	Represents an exception case

Condition page of the transition

A condition is a predicate that applies to a transition. When there are several transitions, the control flow evaluates each condition in order to choose the one it will transit on.

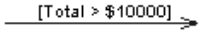
The same concept of condition applies to transitions in statechart and activity diagrams.

Alias

The Alias allows you to edit a condition attached to a transition. It summarizes the condition attached to a transition. You can also use a text editor in which you detail the condition. For example, you can write information on the condition to execute, as well as open, insert and save any text files containing valuable information.

It is recommended that you write an alias (short expression) when you use a long condition so as to display the alias instead of the condition in the diagram.

The condition is displayed near the source symbol as shown below:



Opening the Condition page
You can open the Condition page by right clicking the transition symbol in the diagram, and selecting Condition from the contextual menu.

The concept of condition for an action in the statechart diagram is similar to the concept of condition for transitions.

For more information on the condition of an action, see section Condition page of the action, in chapter Building a statechart diagram.

Creating a transition in an activity diagram

You can create a transition:

- From a diagram
- From the Browser

Transitions name and code are calculated. The name and code boxes display a read-only value named after both extremities of the transition.

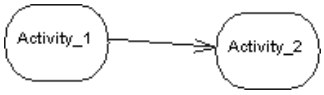
For more information on the different ways to create a transition, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a transition from an activity diagram:**

1 Click the Transition tool.

2 Click inside the first activity (or object state) and while continuing to hold down the mouse button, drag the cursor to the second activity. Release the mouse button inside the second activity.

The transition link appears between the two objects.

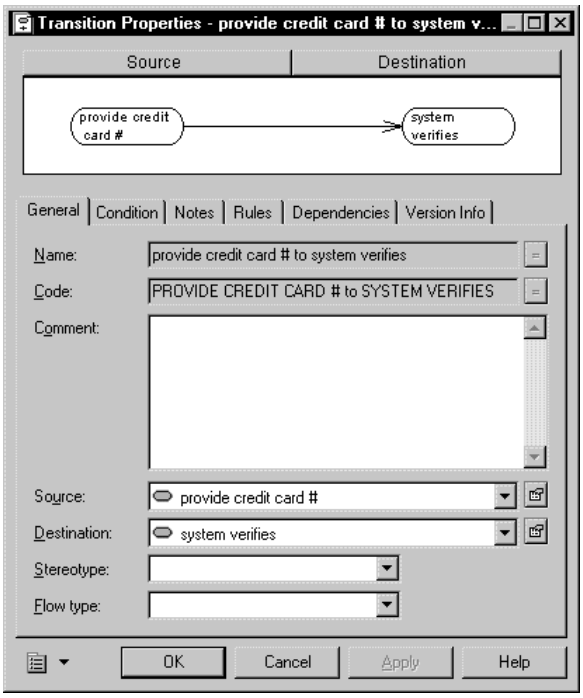


Dragging a transition to a different activity
You change the activity at either end of a transition by clicking the transition to select it, holding the CTRL key down, and dragging one of the attach points to a different activity.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Transition tool.

4 Double-click the transition in the diagram to display its property sheet.



You can open the property sheet of the source and destination objects by clicking the Source and Destination buttons located in the upper part of the transition property sheet.

5 Type or select transition properties as required in the tabbed pages.

6 Click OK.

The transition is also displayed in the property sheet of the activity (or the object state), you can display it from the Input Transitions or Output Transitions tabbed page of the Dependencies page.

Modifying transition properties in an activity diagram

You can modify the transition properties using the following methods:

- From the transition property sheet
- From the list of transitions

For more information on the different ways to modify the transition properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

If your model contains transitions from statechart and activity diagrams, you can filter the transition you need from the List of Transitions. Depending on where you select the command, the content of the list of transitions varies as follows:

Select the command	Displays
Right-click the model in the Browser and select List of→Transitions	All transitions from statechart and activity diagrams
Double-click an activity diagram in the Browser and select Model→Transitions	A typed list of transitions from the activity diagram. The title of the list indicates the diagram type

Modifying transition display preferences in an activity diagram

You can modify the following display preference for transitions using the Tools→Display Preferences command:

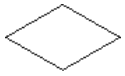
Preference	Description
Show stereotype	Displays the stereotype of the transition

Defining decisions in an activity diagram

A **decision** specifies which alternate path has to be taken when several transition paths are possible. It can have one or more input transitions and one or more output transitions, each labeled with a distinct guard condition. The process of a decision is based on some defined expressions that direct the control flow towards the valid transition by dynamically evaluating guard conditions.

A guard condition is a condition that must be satisfied for an associated transition to execute some action. Across all these transitions, guard conditions should not overlap to avoid ambiguity but they should cover all possibilities to avoid process freeze.

The symbol of a decision is a diamond shape:



Applicability

A decision allows you to create complex flows like:

- if … then … else …
- switch … case …
- do … while …
- loop
- for … next …

Handling symbols in the diagram
It is not possible to attach two transitions of opposite directions to the same corner on a decision symbol.

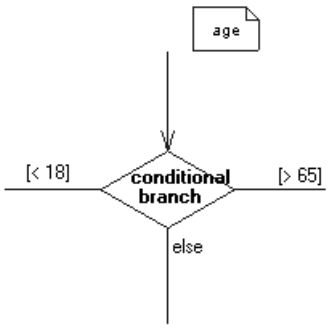
When working with a decision, you can choose to use a **conditional branch** or a **merge**. They describe two different situations:

- A conditional branch is used to define a conditional branching: one input flow, several output flows
- A merge is used to define the opposite situation: several input flows, one output flow

Defining conditional branches

A **conditional branch** joins multiple transitions. It is a place within an activity diagram where a specific event leads to more than one possible outcome, each of which has an associated guard condition or the keyword ELSE.

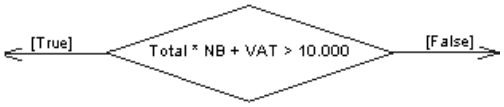
Symbol



Factorizing conditions

If you are working with a conditional branch, it is useful to write a condition on the decision in order to factorize the conditions attached to the transitions, it allows you to simplify the process whenever you are using long and complex conditions.

Example

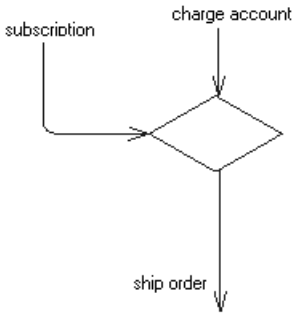


Use the Condition page in the decision property sheet to write the following condition: Total * NB + VAT > 10.000. Then use the Condition page in both transitions property sheet: enter True in one and False in the other.

Defining merges

A **merge** is a scenario within an activity diagram where two or more alternate control paths come together, it represents the merge of several potential transitions into a common path.

Symbol



Decision properties in an activity diagram

You can double-click any decision symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the decision	254
Code	Reference name of the decision	254
Comment	Descriptive comment for the decision	__
Stereotype	Extends the semantics of a decision derived from existing decisions but specific to your needs	254
Type	Calculated read-only value showing the type of the decision	__

Type property of the decision

A decision displays the following types:

When the type is ..	It means that …
Incomplete	No input or no output transition has been defined *or* Exactly one input and one output transitions have been defined
Conditional branch	One input and several outputs have been defined
Merge	Several inputs and one output have been defined

Condition page of the decision

A condition is a predicate that applies to a decision.

Alias

The Alias allows you to edit a condition attached to a decision. It summarizes the condition attached to a decision. You can also use a text editor in which you detail the condition. For example, you may want to write more information on the condition as well as open, insert and save any text files containing valuable information.

It is recommended that you write an alias (short expression) when you use a long condition so as to display the alias instead of the condition in the diagram.

Opening the Condition page
You can open the Condition page by right clicking the decision symbol in the diagram, and selecting Condition from the contextual menu.

Creating a decision in an activity diagram

There are several ways to create a decision:

- From a diagram
- From the Browser
- From the list of decisions

At creation, a decision has a default name including a number, this number is assigned in the order of creation.

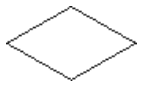
For more information on the different ways to create a decision, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a decision from an activity diagram:**

1 Click the Decision tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



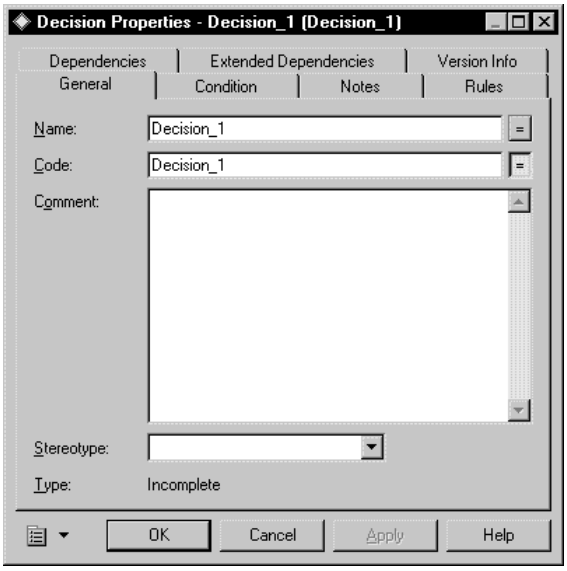
3 Click the Pointer tool.
or
Click the right mouse button.

You release the Decision tool.

4 Double-click the decision symbol to display the decision property sheet.

> **Opening property sheets at last accessed page**
> Property sheets open to the General page by default. However, you can choose to open property sheets at the last page accessed by selecting Tools→General Options→Dialog, and selecting the option Keep Last Tab in the Property Sheets groupbox.

5 Type a name and a code.



6 Click OK.

Modifying decision properties in an activity diagram

You can modify the decision properties using the following methods:

- From the decision property sheet
- From the list of decisions

For more information on the different ways to modify the decision properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying decision display preferences in an activity diagram

You can modify the following display preference for decisions using the Tools→Display Preferences command:

Preference	Description
Show stereotype	Displays the stereotype of the decision

Defining synchronizations in an activity diagram

A **synchronization** is an object that enables synchronization of control between two or more concurrent actions. Its symbol is usually a thick horizontal line, like shown below, but it can also be drawn vertically.

When working with a synchronization, you can choose to either use a **fork** or a **join**. They have different layouts, depending on the number of input and output transitions.

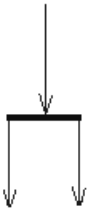
Defining forks

A **fork** is the splitting of an input transition into several output transitions executed in parallel. It creates several concurrent branches below its symbol. The activities associated with each of these paths continue in parallel.

A fork should have only one input flow and have more than one output flow.

It is a complex transition within which one source activity is replaced with two or more destination activities, each of which representing an independent transition (or control flow).

Symbol



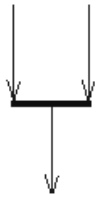
Defining joins

A **join** is the synchronization of several parallel transitions. It is a complex transition with two or more source activities and one destination activity.

A join should have more than one input flow, and have only one output flow.

The activities associated with each of the paths above the join symbol are parallel. In a join, concurrent transitions synchronize. Each transition waits until all input flows reach the join before continuing; the first executed transition waiting for the last one to complete. After the join, only one transition continues below the join.

Symbol



Synchronization properties in an activity diagram

You can double-click any synchronization symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the synchronization	254
Code	Reference name of the synchronization	254
Comment	Descriptive comment for the synchronization	__
Stereotype	Extends the semantics of a synchronization derived from existing synchronizations but specific to your needs	254
Timeout	Defines a timeout limit for waiting until all transitions end. It is empty when the value = 0	__
Type	Calculated read only value showing the pattern of the synchronization	__

Type

A synchronization displays the following types:

When the type is ..	It means that …
Incomplete	No input or no output transitions have been defined *or* Exactly one input and one output transitions have been defined
Fork	One input and several outputs have been defined
Join	Several inputs and one output have been defined

Creating a synchronization in an activity diagram

There are several ways to create a synchronization:

- From a diagram
- From the Browser
- From the list of synchronizations

At creation, a synchronization has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a synchronization, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a synchronization from an activity diagram:**

1 Click the Synchronization tool.

2 Click anywhere in the diagram.

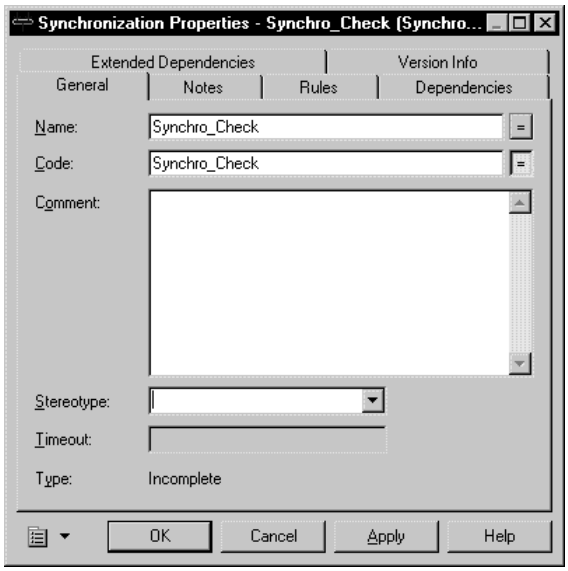
The following symbol appears at the click position:

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Synchronization tool.

4 Double-click the synchronization symbol to display the synchronization property sheet.

5 Type a name and a code.



6 Click OK.

Drawing the synchronization symbol in an activity diagram

The synchronization symbol is created horizontally. However, you may need to move the symbol from a horizontal to a vertical position, and backwards to better fit the layout of your diagram.

Drawing a vertical synchronization symbol in an activity diagram

❖ **To draw a vertical synchronization symbol:**

1 Right-click the synchronization symbol you want to modify.

2 Select Change to Vertical from the contextual menu.

Drawing an horizontal synchronization symbol in an activity diagram

❖ **To draw an horizontal synchronization symbol:**

1 Right-click the synchronization symbol you want to modify.

2 Select Change to Horizontal from the contextual menu.

Modifying synchronization properties in an activity diagram

You can modify the synchronization properties using the following methods:

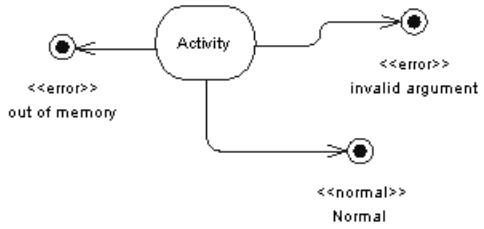
- From the synchronization property sheet
- From the list of synchronizations

For more information on the different ways to modify the synchronization properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Defining ends in an activity diagram

An **end** is a termination point of the activities described in the activity diagram. Its symbol is a solid ball inside a circle (bull's eye).

There can be several ends in the same activity diagram if you want to show divergent end cases, like errors scenarios for example:



There can also be no end at all if you want to show an endless process between two activities.

You are not allowed to use shortcuts of an end.

The end shares the same concept in the statechart diagram and in the activity diagram, that is why you can use the same end simultaneously in a statechart and in an activity diagram.

For more information on ends in the statechart diagram, see section Defining ends in a statechart diagram, in chapter Building a statechart diagram.

End properties in an activity diagram

You can double-click any end symbol in a diagram to display its properties.

Property	Description	Maximum length
Name	Name of the end	254
Code	Reference name of the end	254
Comment	Descriptive comment for the end	__
Stereotype	Extends the semantics of an end derived from existing ends but specific to your needs	254

Creating an end in an activity diagram

There are several ways to create an end:

- From a diagram
- From the Browser
- From the list of ends

At creation, an end has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create an end, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create an end from an activity diagram:**

1 Click the End tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:

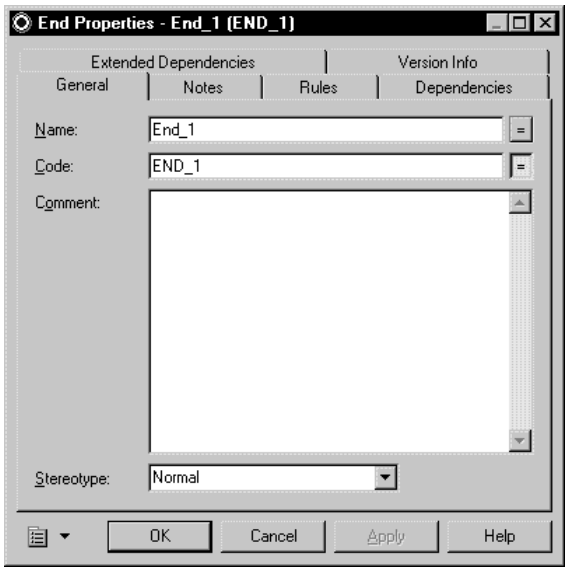


3 Click the Pointer tool.
or
Click the right mouse button.

You release the End tool.

4 Double-click the end symbol to display the end property sheet.

5 Type a name and a code.



6 Click OK.

Modifying end properties in an activity diagram

You can modify the end properties using the following methods:

- From the end property sheet
- From the list of ends

For more information on the different ways to modify the end properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying end display preferences

You can modify the following display preferences for ends using the Tools→Display Preferences command.

The same display preferences apply to starts and ends in statechart and activity diagrams.

Preference	Description
Show name	Displays the name of the end
Show stereotype	Displays the stereotype of the end

CHAPTER 9

Building a Component Diagram

About this chapter

This chapter describes how to build a component diagram. It explains the role of each element in a component diagram and how to create and modify them.

Contents

Topic	Page
Component diagram basics	354
Defining components	356
Defining generalizations in a component diagram	370
Defining dependencies in a component diagram	374

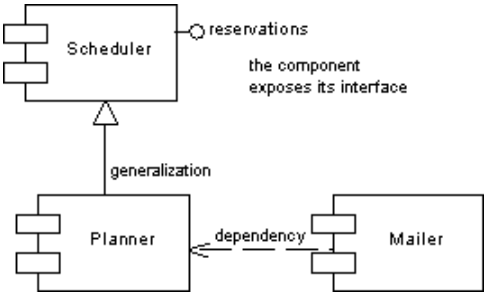
Component diagram basics

The **component diagram** is part of the Unified Modeling Language (UML). It shows dependencies among software components.

Defining a component diagram

The component diagram is part of an Object-Oriented Model. It is one of the available diagrams in an OOM. It shows the symbols of components defined in the model including interfaces defined in components and the relationships between them.

It addresses the static implementation view of a system, drawn as a graph of software components connected by dependencies. Components may also be connected by generalization relationships when a component inherits from another one.



Building a component normally comprises the following phases: design, develop, update components, then from the server side the ability to deploy, catalog and find components.

You can create other types of components if you are working with the Java or .NET families. EJB, servlets, JSP and ASP.NET components are introduced and defined in a specific chapter.

For more information on EJB, servlets, JSP and ASP.NET, see chapter Working with platform dependent components.

Why build a component diagram?

Component diagrams are used to define object dependencies and relationships at a higher level than class diagrams.

You use the component diagram to model the structure of the software, and show dependencies among source code, binary code and executable components so that the impact of a change can be evaluated. You also use the component diagram to show components that abide by the contract with their associated interfaces.

Components should be designed in order to be reused for several applications, they should be extended without breaking existing applications. Therefore the design of components becomes an important task; their definition, creation and deployment should be thought out carefully.

A component diagram is useful during analysis and design. It allows analysts and project leaders to specify the components they need before having them developed and implemented. The component diagram provides a view of components and makes it easier to design, develop, and maintain components and help the server to deploy, catalog, and find components.

For more information on diagrams of the OOM, see sections What is an OOM, in chapter Object-Oriented Model Basics.

Creating a component diagram

You can create a component diagram using different methods.

For more information on creating a component diagram in a new OOM, see section Creating an OOM, in chapter Object-Oriented Model Basics.

For more information on creating a component diagram in an existing OOM, see section Creating a new diagram, in chapter Managing Models, in the *General Features Guide*.

Defining components

A **component** is a physical, replaceable part of a system that packages implementation, conforms to and provides the realization of a set of interfaces. It can represent a physical piece of implementation of a system, like software code (source, binary or executable), scripts, or command files. It is an independent piece of software developed for a specific purpose but not a specific application. It may be built up from the class diagram and written from scratch for the new system, or it may be brought in from other projects and third party vendors.

The symbol of the component is the following:



A component provides a 'black box' building block approach to software construction. For example, from the outside, a component would show two interfaces that describe it, whereas from the inside, the component would reflect both interfaces realized by a class, both operations of the interfaces being the operations of the class.

A component developer has an internal view of the component: its interfaces and implementation classes, whereas one who assembles components to build another component or an application only has the external view (the interfaces) of these components.

A component can be implemented in any language. In Java, you can implement EJB, servlets, and JSP components, for example.

For more information on other types of components: EJB, servlets, JSP and ASP.NET, see chapter Working with platform dependent components.

If you start developing a component with classes and interfaces in an OOM and you later want to store them in a database, it is possible to create a manual mapping of objects so that OOM objects correspond to PDM objects. Similarly, if you have an existing OOM and an existing PDM and both models must be preserved; you can handle the link between the object-oriented environment and the physical database through the **object to relational mapping**. Using this mapping, you can make your components communicate to each other and evolve in an object environment, as well as retrieve data stored in a database.

For more information on O/R Mapping, see section Understanding O/R mapping, in chapter Working with Object-Oriented Models.

Component properties

You can double-click any component symbol in a diagram to display its properties.

A component has the following general properties:

Property	Description	Maximum length
Name	Name of the component	254
Code	Reference name or code for the component	254
Comment	Descriptive comment for the component	—
Stereotype	Extends the semantics of a component derived from existing components but specific to your needs. It can be predefined or user-defined	—
Type	Defines the type of component	—
Transaction	Used for a component with transactional behavior	—
Class diagram	Diagram with classes and interfaces linked to the component. It is automatically created and updated	—
Web service	Indicates that the component is a Web service	—

For more information on the class diagram property, see Creating a class diagram for a component.

Component stereotype

You can declare a component to be one of the following default stereotypes:

Stereotype	Description
<<Document>>	Generic file that is not a source file or an executable
<<Executable>>	Program file that can be executed on a computer system
<<File>>	Physical file in the context of the system developed
<<Library>>	Static or dynamic library file
<<Table>>	Database table

You can modify an existing stereotype or create a new one from an object language. You can also use the extended model definition to create new sets of stereotypes.

For more information on how to manipulate categories and entries in the object language, see chapter The Resource Editor in the *General Features Guide*.

For more information on extended model definitions, see section Working with extended model definitions, in chapter Managing Objects in the *General Features Guide*.

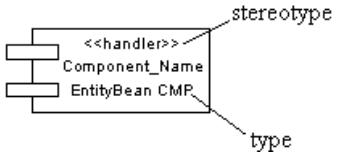
Component type

The component type defines the characteristics of the component. You can either use a standard or a specific component. A standard component is useful if no specific implementation of the component has been defined. A specific component includes EJB, JSP, Servlet or ASP.NET.

For more information on specific component types, see chapter Working with platform dependent components.

The type of a component can be displayed in the component symbol in the diagram.

To display the type of a component, select Tools→Display Preferences and select the Show Type option in the component category.



Whenever you change the type of a component after creation, the modification triggers a conversion from one type to another: all relevant interfaces and classes together with all dependencies are automatically created and initialized. Such a change will automatically affect some property sheets as well as the Check Model feature, and code generation.

Example

If you convert a standard component to an EJB Entity Bean, it will automatically generate a Bean class and a primary key class of the EJB, as well as home and component interfaces. If you happen to convert an EJB to a standard component, the classes and interfaces of the EJB are preserved in the model.

Component related diagrams

In the component property sheet, you can select the Related Diagrams tab to display all diagrams of the model that are related to a component. They can be attached and detached from the current component at this stage. This page displays a list of the diagrams identified by their name and code.

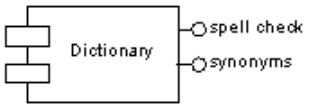
For more information on Related Diagrams, see section Defining related diagrams, in chapter Working with Object-Oriented models.

Defining interfaces for a component

Interfaces contained in the component are interfaces exposed by the component. This indicates that the component implements the services provided by the interface.

Each component uses one or several interfaces. It also uses or requires interfaces from other components. These interfaces are visible entry points and services that a component makes available to other software components and classes. If dependencies among components originate from interfaces, these components can be replaced by other components that use the same interfaces.

Component interfaces are shown as circles linked to the component side by an horizontal or a vertical line:



The symbol of a component interface is visible if you have selected the Show Interface symbols display preference from Tools→Display Preferences. The symbol of an interface can be moved around the component symbol, and the link from the component to the interface can be extended.

Displaying the interface properties
You can double-click the circle, symbol of an interface to display the interface property sheet.

If you are working with EJB, some of the interfaces have a special meaning (local interface, remote interface, etc…).

For more information on EJB interfaces, see section Defining interfaces and classes for EJB in chapter Working with platform dependent components.

Interfaces of a component can be in the same model or in another model. If they are in another model, you can use external shortcuts of interfaces.

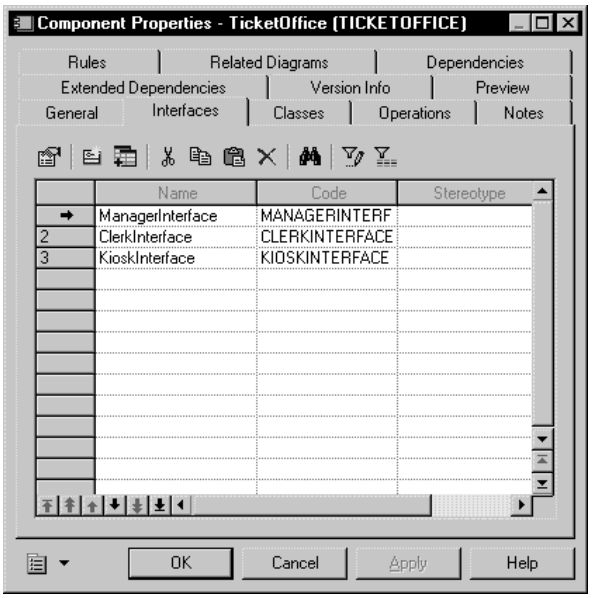
You can define an interface for a component from the Interfaces page of the component property sheet.

❖ **To define interfaces for a component:**

1 Double-click the component in the diagram to display the component property sheet.

2 Click the Interfaces tab to display the Interfaces page.

3 Click the Create a New Interface tool to create a new interface. The interface property sheet appears.
Type a name and a code and click OK.
or
Click the Add Objects tool to display a Selection window.
Select the interfaces you want to attach to the component and click OK.



The interfaces appear in the list, and their definitions are added to the current component definition. Interfaces are created in the same package as the component.

Defining classes for a component

Classes contained in the component are classes implementing the component.

A component has one or several classes. It usually uses one implementation class as the main class, while other classes are used to implement the functions of the component. Typically, a component consists of many internal classes and packages of classes but it may also be assembled from a collection of smaller components.

These classes have no symbol in the component diagram, unlike interfaces. However, the relationship between the class and the component is visible from the Classes page of the component property sheet, and from the Components tabbed page in the Dependencies page of the class property sheet.

Classes of a component can be in the same model or in another model. If they are in another model, you can use external shortcuts of classes.

You can define a class for a component from the Classes page of the component property sheet.

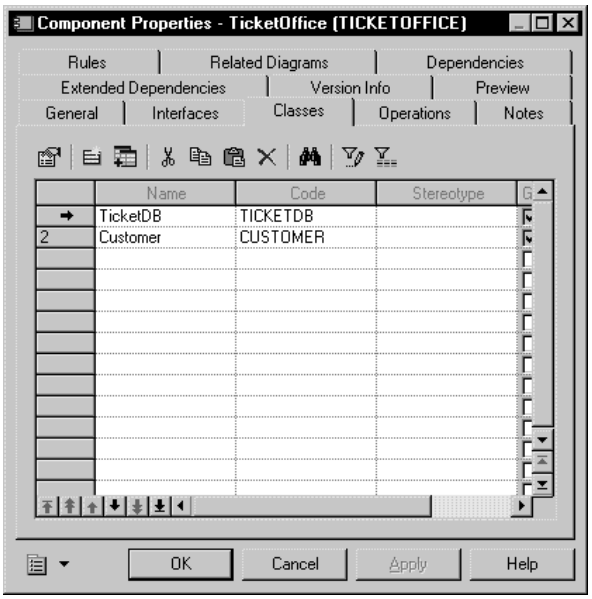
❖ **To define a class for a component:**

1 Double-click the component in the diagram to display the component property sheet.

2 Click the Classes tab to display the Classes page.

3 Click the Create a New Class tool to create a new class. The class property sheet appears.
Type a name and a code and click OK.
or
Click the Add Objects tool to display a Selection window.

Select the classes you want to attach to the component and click OK.



The classes appear in the list, and their definitions are added to the current component definition. Classes are created in the same package as the component.

Defining file objects for a component

You can define file objects to be attached to a component from the Files page in the component property sheet.

This file object is a file containing text or images. It can be attached to a component that you will later deploy into a server. The advantage of this is that the information contained within the file object travels with the component, and it is deployed at the same time as the component.

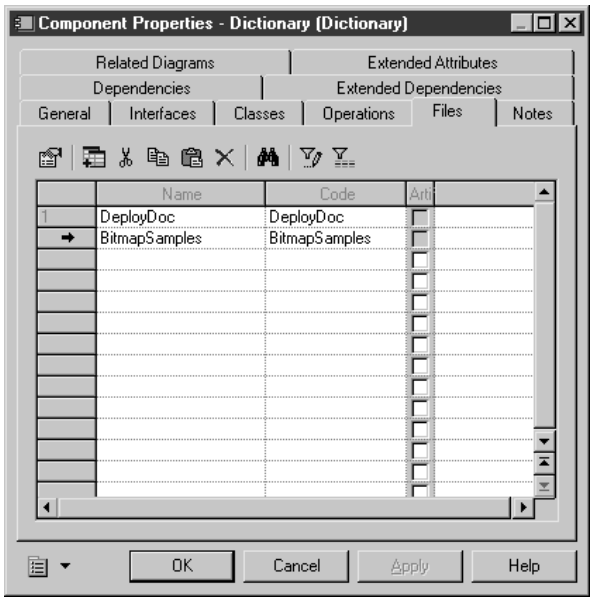
❖ **To define a file object for a component:**

1 Double-click the component in the diagram to display the component property sheet.

2 Click the Files tab to display the Files page.

3 Click the Add Objects tool to display a Selection window.

4 Select the file object you want to attach to the component and click OK.



The name of the file object appears in the list.

5 Click OK to close the component property sheet.

Viewing operations of a component

Operations contained in the component are operations declared in the component interfaces.

You can select a specific interface, or you can select all interfaces of the component and display their properties from the dropdown listbox of the Operations page in the component property sheet.

It is not possible however to create an operation from the Operations page of the component property sheet, this page is only used to view the operations of the component. You need to create an operation directly from the interface property sheet.

❖ **To view operations of a component:**

1 Double-click the component in the diagram to display the component property sheet.

2 Click the Operations tab to display the Operations page.

3 Select one interface or <All interfaces> from the dropdown listbox.

The operations of the interface appear in the list. You can double-click an arrow at the beginning of a line to display an operation property sheet.

Creating a class diagram for a component

You can create a class diagram for a selected component to have an overall view of the classes and interfaces associated with the component. You can only create one class diagram per component.

The Create/Update Class Diagram feature from the component contextual menu, acts as follows:

- Creates a class diagram if none exists
- Attaches a class diagram to a component
- Adds symbols of all related interfaces and classes in the class diagram
- Completes the links in the diagram

This feature also allows you to update a class diagram after you have made some modifications to a component.

Opening a class diagram

The Open Class Diagram feature, available from the component contextual menu, opens the specific class diagram if it exists, or it creates a new default class diagram.

For EJB components for example, the Open Class Diagram feature opens the class diagram where the Bean class of the component is defined.

If you delete a component that is attached to a class diagram, the class diagram is also deleted. Moreover, the classes and interfaces symbols are deleted in the class diagram, but the classes and interfaces objects remain in the model.

❖ **To create a class diagram for a component:**

- Right-click the component in the component diagram and select Create/Update Class Diagram from the contextual menu.

  A new class diagram, specific to the component, is displayed in the diagram window and the corresponding node appears under in the Browser. You can further create objects related to the component in the new class diagram.

Opening the class diagram
To open the class diagram for a component, right-click the component in the diagram and select Open Class Diagram from the contextual menu or press CTRL and double-click the component.

Creating a component

There are several ways to create a standard component:

- From a class diagram
- From a component diagram
- From the list of components

At creation, a standard component has a default name including a number, this number is assigned in the order of creation. The creation of a standard component is not depending on the language you are using.

For more information on the different ways to create a component, see section Creating an object in chapter Managing objects in the *General Features Guide*.

For more information on other types of components: EJB, servlets, JSP and ASP.NET, see chapter Working with platform dependent components.

Creating a component from a class diagram

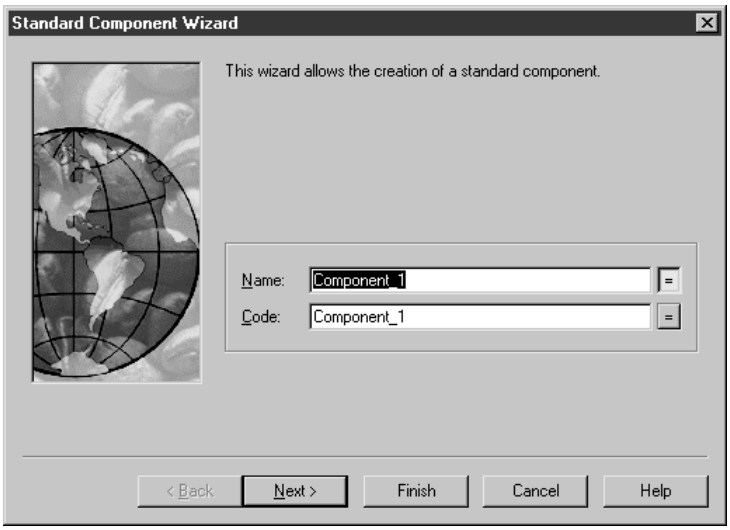
When you create a component from a class diagram, you have to define the following options:

Option	Description
Name	Name of the component
Code	Code of the component
Create symbol	Creates a component symbol. It is created in the diagram specified beside the Create symbol In check box
Create Class Diagram for component classifiers	Creates a class diagram with a symbol for each class and interface. If you have selected classes and interfaces before starting the wizard, they are attached to the component. This option allows you to display these classes and interfaces

❖ **To create a component from a class diagram:**

1 Select Tools→Create Component from a class diagram.

The Standard Component Wizard dialog box appears.



2 Type a name and a code for the component and click Next.

3 Select the Create Symbol In check box to create a component symbol. The diagram you select in this dialog box is the diagram that contains the component symbol.

If the Create Symbol In check box is not selected, the component is created and visible from the Browser but it has no symbol.

If you have selected classes or interfaces before using the wizard, classes and interfaces are automatically attached to the component.

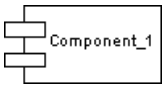
Creating a component from a component diagram

❖ **To create a component from a component diagram:**

1 Click the Component tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:

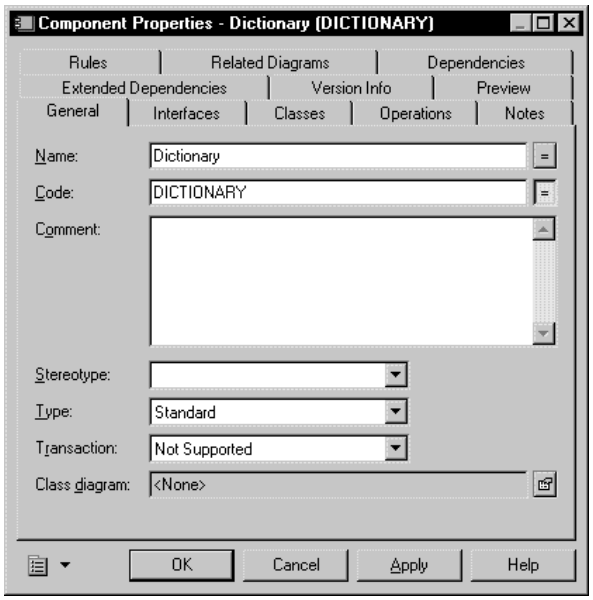


3 Click the Pointer tool.
or
Click the right mouse button.

You release the Component tool.

4 Double-click the component symbol to display the component property sheet.

5 Type a name and a code.



6 Click OK.

Deploying a component to a node

Deploying a component to a node allows you to set an instance of the component within a node. You can deploy a component from the component diagram or from the Browser. After you deploy the component, a shortcut of this component and a new component instance are created within the deployment node.

You can only select one component to deploy at a time.

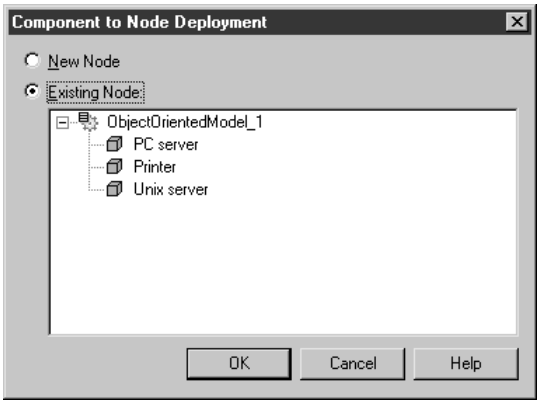
❖ **To deploy a component to a node:**

1 Right-click the component symbol.

2 Select Deploy Component to Node from the contextual menu.

The Component to Node Deployment window appears.

3 Select New Node to create a new node.
or
Select an existing node.



4 Click OK.

The node includes a new component instance that refers to the current component.

Modifying component properties

You can modify the component properties using the following methods:

- From the component property sheet
- From the List of Components

For more information on the different ways to modify the component properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying component display preferences

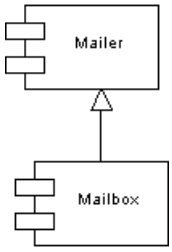
You can modify the following display preferences for components using the Tools→Display Preferences command:

Preference	Description
Show stereotype	Displays the stereotype of the component
Show type	Displays the type of the component
Show interface symbols	Displays the symbols of interfaces exposed by the component

Defining generalizations in a component diagram

A **generalization** is a relationship between a more general element (the parent) and a more specific element (the child). The more specific element is fully consistent with the more general element and contains additional information.

A generalization can be created between two components, as shown below:



Generalization properties in a component diagram

You can double-click the generalization symbol in a diagram to display its properties.

A generalization has the following properties:

Property	Description	Maximum length
Name	Name of the generalization	254
Code	Reference name of the generalization	254
Comment	Descriptive comment for the generalization	—
Parent	Parent class or interface of the generalization	254
Child	Child class or interface of the generalization	254
Stereotype	Extends the semantics of a generalization derived from existing generalizations but specific to your needs. <<implementation>> means that the child object inherits the implementation of the parent object but does not make its interfaces public, nor support them, thus violating its substitutability	—
Visibility	Visibility of the generalization, whose value denotes how it is seen outside its enclosing namespace	—

Visibility

Property	Visible
Private	Only to the generalization itself
Protected	Only to the generalization and its inherited objects
Package	To all objects contained within the same package
Public	To all objects

Creating a generalization in a component diagram

There are several ways to create a generalization:

- From a diagram
- From the Browser

At creation, a generalization has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a generalization, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a generalization from a component diagram:**

1 Click the Generalization tool.

2 Click inside a component and while continuing to hold down the mouse button, drag the cursor to another component. Release the mouse button inside the second component.

The generalization link appears between both objects.

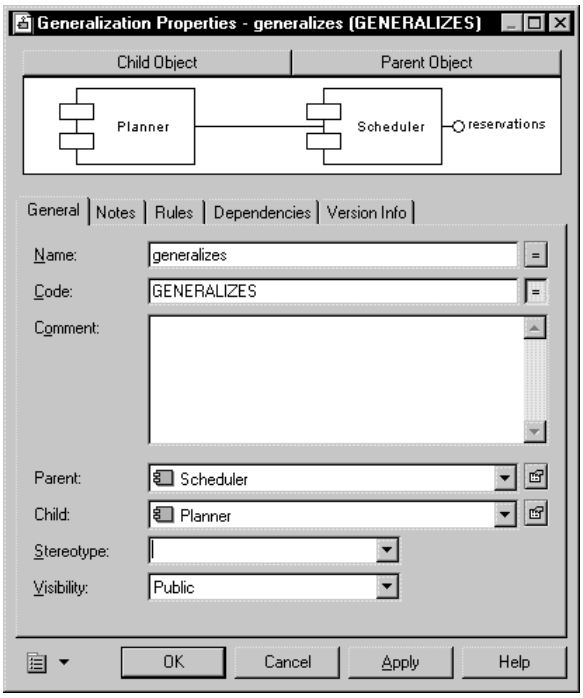
> **Dragging a generalization to a different object**
> You move the generalization to a different object by clicking the generalization line to select it, pressing down CTRL, and dragging one of the attach points to a different object.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Generalization tool.

4 Double-click the generalization to display its property sheet.

5 Type a name and a code.



You can open the property sheet of the child and parent objects by clicking the Child Object and Parent Object buttons located in the upper part of the generalization property sheet.

6 Click OK.

Modifying generalization properties in a component diagram

You can modify the generalization properties using the following methods:

- From the generalization property sheet
- From the list of generalizations

For more information on the different ways to modify the generalization properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying generalization display preferences

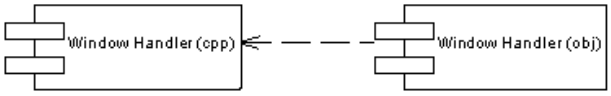
You can modify the following display preferences for generalizations using the Tools→Display Preferences command.

Preference	When selected
Show name	Displays the name of the generalization
Show stereotypes	Displays the stereotypes of the generalization
Show constraints	Displays the constraints (business rules) of the generalization

Defining dependencies in a component diagram

A dependency is a relationship between two modeling elements, in which a change to one modeling element (the influent element) will affect the other modeling element (the dependent element).

A dependency can be created between two components as shown below. You can **not** create a dependency between a component and an interface.



When using a dependency, you can nest two components by using a stereotype.

Dependency properties in a component diagram

You can double-click any dependency symbol in a diagram to display its properties.

A dependency has the following properties:

Property	Description	Maximum length
Name	Name of the dependency	254
Code	Reference name of the dependency	254
Comment	Descriptive comment for the dependency	—
Influent	Component that influences a dependent component. Changes on the influent component affect the dependent one	254
Dependent	Component that depends on an influent component. Changes on the dependent component will not affect the influent one	254
Stereotype	Extends the semantics of a dependency derived from existing dependencies but specific to your needs	—

A dependency has the following default stereotypes in Java:

Stereotype	Description
<<ejb-ref>>	Used in Java Generation to create references to EJBs (entity beans and session beans) for generating the deployment descriptor
<<sameFile>>	Used in Java Generation to generate Java classes of visibility protected or private within a file corresponding to a class of visibility public

Creating a dependency in a component diagram

You create a dependency between components:

- From a diagram
- From the Browser

At creation, a dependency has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a dependency, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ To create a dependency from a component diagram:

1 Click the Dependency tool.

2 Click inside a component and while continuing to hold down the mouse button, drag the cursor to another component. Release the mouse button inside the second component.

The dependency link appears between both objects.

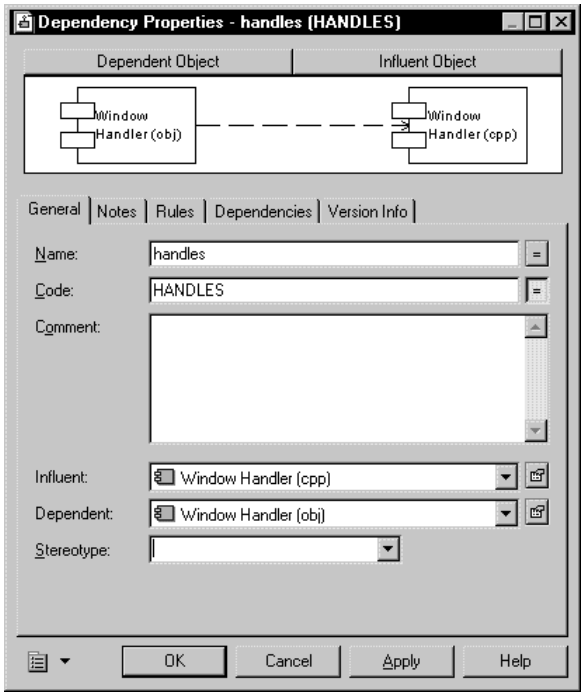
> **Dragging a dependency to a different object**
> You move the dependency to a different object by clicking the dependency line to select it, pressing down CTRL, and dragging one of the attach points to a different object.

3 Click the Pointer tool.
or
Click the right mouse button.

You release the Dependency tool.

4 Double-click the dependency to display its property sheet.

5 Type a name and a code.



You can open the property sheet of the dependent and influent objects by clicking the Dependent Object and Influent Object buttons located in the upper part of the dependency property sheet.

6 Click OK.

Modifying dependency properties in a component diagram

You can modify the dependency properties using the following methods:

- From the dependency property sheet
- From the list of dependencies

For more information on the different ways to modify the dependency properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying dependency display preferences

You can modify the following display preferences for dependencies using the Tools→Display Preferences command.

Preference	When selected
Show name	Displays the name of the dependency
Show stereotype	Displays the stereotype of the dependency
Show constraints	Displays the constraints (business rules) of the dependency

CHAPTER 10

Building a Deployment Diagram

About this chapter

This chapter describes how to build a deployment diagram. It explains the role of each element in a deployment diagram and how to create and modify them.

Contents

Topic	Page
Deployment diagram basics	380
Defining nodes	382
Defining component instances	388
Defining file objects in a deployment diagram	392
Defining node associations	396
Defining dependencies in a deployment diagram	401

Deployment diagram basics

The **deployment diagram** shows the physical configuration of run-time processing elements (nodes) on an instance level. The nodes contain instances of component that will be deployed into database, application or web servers.

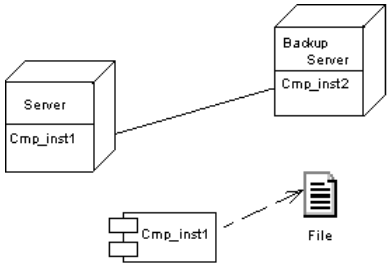
Defining a deployment diagram

The deployment diagram is one of the available diagrams in an OOM (Object-Oriented Model).

The deployment diagram is part of the Unified Modeling Language (UML). It is an implementation diagram that complements the component diagram by giving more accurate details about the physical implementation and interactions of components.

The deployment diagram is a view of nodes connected by communication links. It allows you to design nodes, component instances that can be contained within the node symbol, file objects associated with nodes that are used for deployment, and relationships between nodes.

When the node contains component instances, this indicates that the component executes on the node.



Why build a deployment diagram?

Deployment diagrams are used for actual deployment of components into servers. A deployment can represent the ability to use instances.

You use the deployment diagram to establish the link to the physical architecture. It is suitable for modeling network topologies, for instance.

You can build a deployment diagram to show the following views, from a high level architecture that describes the material resources and the distribution of the software in these resources, to final complete deployment into a server:

- Identify the system architecture: use nodes and node associations only
- Identify the link between software and hardware: use component instances, split up their route, identify and select the servers
- Deploy components into the servers: include some details, add physical parameters

For more information on diagrams of the OOM, see section What is an OOM, in chapter Object-Oriented Model Basics.

Creating a deployment diagram

You can create a deployment diagram using different methods.

For more information on creating a deployment diagram in a new OOM, see section Creating an OOM, in chapter Object-Oriented Model Basics.

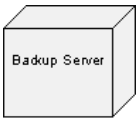
For more information on creating a deployment diagram in an existing OOM, see section Creating a new diagram, in chapter Managing Models, in the *General Features Guide*.

Defining nodes

A node is the main element of the deployment diagram. It is a physical element that represents a processing resource, a real physical unit, or physical location of a deployment (computer, printer, or other hardware units).

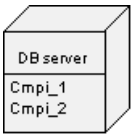
In UML, a node is defined as Type or Instance. This allows you to define for example 'BackupMachine' as node Type, and 'Server:BackupMachine' as Instance. As a matter of simplification, PowerDesigner handles only one element, called node, which actually represents a node instance. If you need to designate the type, you can use a stereotype for example.

The symbol of a node is a cube:



A node **cannot** contain another node, however it can contain component instances and file objects: the software component instances and/or associated file objects that are executed within the nodes. You can use shortcuts of component as well.

You can add a component instance from the node property sheet. You can display the list of component instances in the node symbol as well, by selecting the option Show components in the node display preferences.



Defining node diagrams

You can expand a node in a diagram owned by the node, in order to define its contents: the associated component instances and its file objects.

This diagram is a deployment diagram within a node. It is called a **node diagram**.

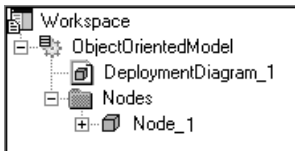
Creating a node diagram

You create a node diagram from the symbol of the parent node.

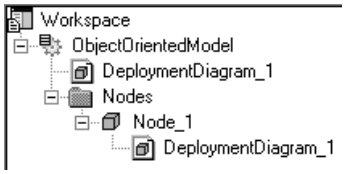
❖ **To create a node diagram:**

♦ Press CTRL and double-click the parent node symbol from the deployment diagram.

A default deployment diagram opens in the diagram window. The node of the parent node is marked with a + sign in the Nodes category of the Browser.



If you expand the node of the parent node, you see that a deployment diagram is automatically created. It is the node diagram in which you can create any component instances and file objects associated with the node.



Opening a node diagram

You can open a node diagram using the following methods:

♦ Press CTRL and double-click on the parent node symbol.
or
Select Open Diagram from the contextual menu of the parent node.

The node diagram opens in the diagram window.

Closing a node diagram

You can close a node diagram using the following methods:

♦ Press CTRL + U in the node diagram to go up to the parent diagram.
or
Select Diagram→Go Up One Level from the contextual menu.

Node properties

You can double-click any node symbol in a diagram to display its properties:

Property	Description	Maximum length
Name	Name of the node	254
Code	Reference name or code for the node	254
Comment	Descriptive comment for the node	—
Stereotype	Extends the semantics of a node derived from existing nodes but specific to your needs	254
Cardinality	Specific numbers of instances that the node can have, for example: 0...1	—
Network address	Address or machine name	254

Component instances page of the node

The node property sheet includes a Component Instances page that lists all instances of components that can run or execute on the current node. It allows you to define component instances directly from the node definition without any drill down into the node diagram. Once defined, the component instances can be displayed within the node symbol, provided the Show Component Instances option is selected in the node display preferences.

Opening the Component Instances page
You can open the Component Instances page by right clicking the node symbol in the diagram, and selecting Component Instances from the contextual menu.

For more information on component instances, see section Defining component instances.

Creating a node

There are several ways to create a node:

- From a diagram
- From the List of Nodes

At creation, a node has a default name including a number, this number is assigned in the order of creation.

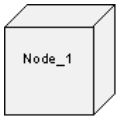
For more information on the different ways to create a node, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a node from a diagram:**

1 Click the Node tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



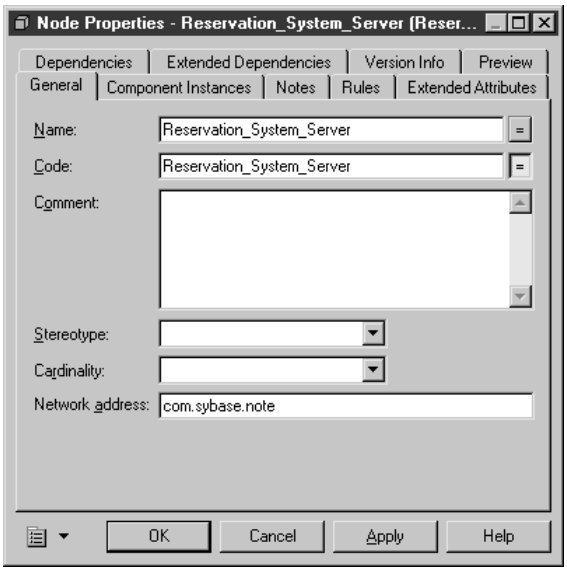
3 Click the Pointer tool.
or
Click the right mouse button.

You release the Node tool.

4 Double-click the node symbol to display its property sheet.

Opening property sheets at last accessed page
Property sheets open to the General page by default. However, you can choose to open property sheets at the last page accessed by selecting Tools→General Options→Dialog, and selecting the option Keep Last Tab in the Property Sheets groupbox.

5 Type a name and a code.



6 Click OK.

The node item appears in the Node category of the Browser.

Modifying node properties

You can modify the node properties using the following methods:

- From the node property sheet
- From the List of Nodes

For more information on the different ways to modify the node properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying node display preferences

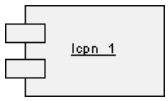
You can modify the following display preferences for nodes using the Tools→Display Preferences command:

Preference	Description
Show component instances	Displays the component instances defined for the node in the node symbol
Show stereotype	Displays the stereotype of the node
Show network address	Show node address

Defining component instances

A component instance is an instance of a component that can run or execute on a node. Whenever a component is processed into a node, a component instance is created. The component instance plays an important role because it contains the parameter for deployment into a server.

The component instance symbol is the same as the component symbol in the component diagram.



The component instance relationship with the node is similar to a composition; it is a strong relationship, whereas the file object relationship with the node is different because several nodes can use the same file object according to deployment needs.

Drag and drop a component in a deployment diagram

You can drag a component from the Browser and drop it into a deployment diagram to automatically create a component instance linked to the component.

The component instance that inherits from the component automatically inherits its type: the type of the component is displayed in the property sheet of the component instance.

Deploy component to node from the component diagram

You can create a component instance from a component. To do this, use the Deploy Component to Node feature. This feature is available from the contextual menu of a component (in the component diagram) or from the Browser. This creates a component instance and attaches the component instance to a node. If you display the node symbol in a deployment diagram, the component instance name appears within the node symbol to which it is attached.

For more information on deploying a component to a node, see section Deploying a component to a node, in chapter Building a component diagram.

Component instance properties

You can double-click any component instance symbol in a diagram to display its properties:

Property	Description	Maximum length
Name	Name of the component instance	254
Code	Reference name or code for the component instance	254
Comment	Descriptive comment for the component instance	—
Stereotype	Extends the semantics of a component instance derived from existing component instances but specific to your needs	254
Cardinality	Specific number of occurrences that the component instance can have, for example: 0...1	—
Component	Component of which the component instance is an instance. If you change the component name in this box, the name of the component instance is updated in the model	254
Component type	Read-only box that shows the type of the component from which the component instance is coming	254
Web service	Indicates that the component instance is an instance of a Web service component	—

If you want to list all component instances of a component, click the Component Instances tabbed page in the Dependencies page of the component property sheet.

Creating a component instance

There are several ways to create a component instance:

- From a diagram
- From the Browser

At creation, a component instance has a default name including a number, this number is assigned in the order of creation.

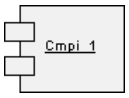
For more information on the different ways to create a component instance, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ To create a component instance from a diagram:

1 Click the Component Instance tool.

2 Click anywhere in the diagram.

The following symbol appears at the click position:

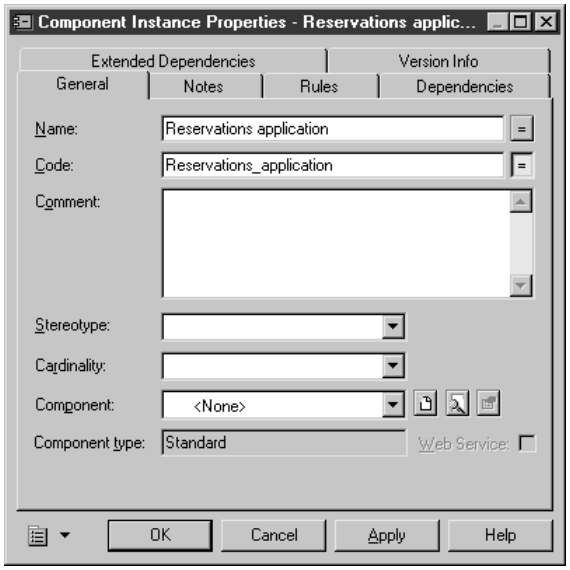


3 Click the Pointer tool.
or
Click the right mouse button.

You release the Component Instance tool.

4 Double-click the component instance symbol to display its property sheet.

5 Type a name and a code.



6 Click OK.

The component instance item appears in the Component Instances category of the Browser.

Modifying component instance properties

You can modify the component instance properties using the following methods:

- From the component instance property sheet
- From the List of Component Instances

For more information on the different ways to modify the component instance properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying component instance display preferences

You can modify the following display preferences for component instances using the Tools→Display Preferences command:

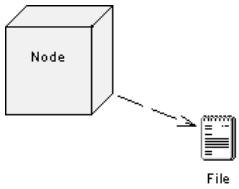
Preference	Description
Show stereotype	Displays the stereotype of the component instance
Show type	Displays the type of the component from which the component instance is an instance

Defining file objects in a deployment diagram

A file object can be a bitmap file used for documentation, or it can be a file containing text that is used for deployment into a server.



When you want to associate a file object to a node, you can drag a dependency from the file object to the node:



You can also use CTRL and double-click on the parent node symbol, then create the file object into the node diagram.

For more information on node diagrams, see section Defining node diagrams.

You can edit a file object by right-clicking its symbol in the deployment diagram and selecting Open Document or Open With→*text editor* of your choice from the contextual menu.

For more information on file objects, see section Using file objects in chapter Managing objects in the *General Features Guide*.

File object properties in a deployment diagram

The existing file object is extended in order to support the Artifact property. This property allows you to differentiate between file objects that are purely for documentation purpose and artifact file objects that are generated during the generation process.

A file object includes the following properties specific to deployment:

Property	Description
Generate	The file object is automatically included among the objects generated from the model when you launch a generation process
Artifact	The code of an artifact file object is generated as part of the generation process. Only file objects that have the Artifact property selected are generated. Their contents may be edited using the editor of your choice. An artifact file object can be edited in the internal editor of PowerDesigner if its extension corresponds to an extension defined in the Editors page of the General Options dialog box (Tools menu), and if the <internal> keyword is defined in the Editor Name and Editor Command columns for this extension. In this case the Contents tab appears in the file object property sheet, it allows you to directly edit the content of the artifact file object

Creating a file object in a deployment diagram

There are several ways to create a file object:

- From a diagram
- From the Browser

At creation, a file object is embedded by default within the model and has a default name including a number. This number is assigned in the order of creation of objects.

For more information on the different ways to create a file object, see section Creating an object in chapter Managing objects in the *General Features Guide*.

Drag and drop from the Windows Explorer
You can easily create an external file object within your OOM by dragging and dropping a file from the Windows Explorer to your OOM.

❖ **To create a file object from a deployment diagram:**

1 Click the File tool in the Palette.

2 Click anywhere in the diagram.

The following symbol appears at the click position:



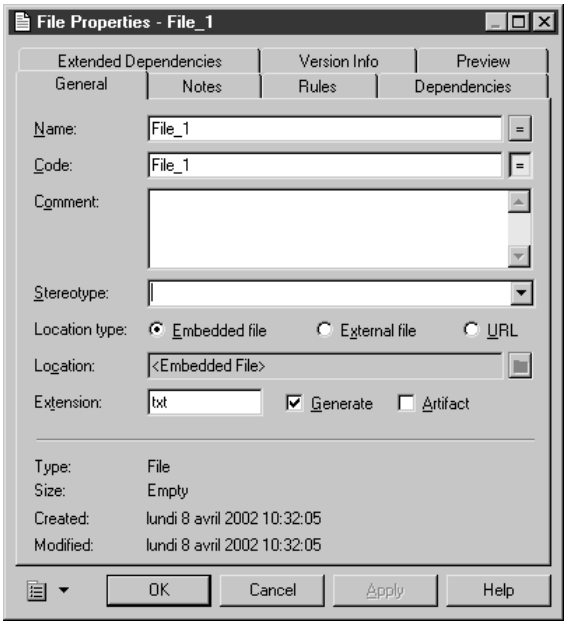
3 Click the Pointer tool.
or
Click the right mouse button.

You release the File tool.

4 Right-click the new file object symbol in the diagram and select properties from the contextual menu to display the file object property sheet.

5 Type a file object name.

6 Select a Location Type.



7 Browse to the directory that contains the external file in the Location box.
or
Type the Web address to access the file on a Web site.

8 Type the file object extension.

9 Select the Generate checkbox if you want to include the file object during generation.

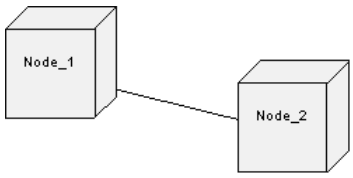
10 Select the Artifact checkbox if you want to generate the code of the file object during generation.

11 Click OK.

For more information on file objects, see section Using file objects in chapter Managing objects in the *General Features Guide*.

Defining node associations

You can create associations between nodes, called node associations. They are defined with a role name and a multiplicity at each end. An association between two nodes means that the nodes communicate with each other, for example when a server is sending data to a backup server.



Node association properties

You can double-click any node association symbol in a diagram to display its properties:

A node association has the following general properties:

Property	Description	Maximum length
Name	Name of the node association	254
Code	Reference name of the node association	254
Comment	Descriptive comment for the node association	—
Stereotype	Extends the semantics of a node association derived from existing node associations but specific to your needs	254
Role A	One side of a node association. Each role can have a name and a cardinality and be navigable	254
Node A	Name of the node at one end of the node association	254
Multiplicity A	Minimum and maximum number of instances of the node association	—
Role B	One side of a node association. Each role can have a name and a cardinality and be navigable	254
Node B	Name of the node at the other end of the node association	254
Multiplicity B	Minimum and maximum number of instances of the node association	—

Node association multiplicity

The allowable cardinalities of a role are called multiplicity. Multiplicity indicates the minimum and maximum cardinality that a role can have.

Multiplicity	Number of instances
0..1	Zero or one
0..*	None to unlimited
1..1	Exactly one
1..*	One to unlimited
*	None to unlimited

The Multiplicity is related to instances of a node association. For example, in a computer environment, there can be 100 clients and 100 machines but there is a constraint that says that a machine can accept at most 4 clients at the same time. In this case, the maximum number of instances is set to 4 in the Multiplicity box on the machine side:



Creating a node association

You create a node association between two nodes:

- From a diagram
- From the Browser

At creation, a node association has a default name including a number, this number is assigned in the order of creation.

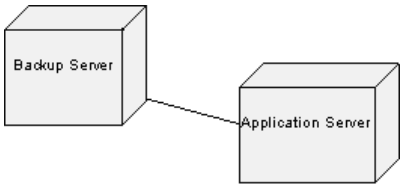
For more information on the different ways to create a node association, see section Creating an object in chapter Managing objects in the *General Features Guide*.

❖ **To create a node association from a diagram:**

1 Click the Node Association tool.

2 Click inside the first node and while continuing to hold down the mouse button, drag the cursor to the second node. Release the mouse button inside the second node.

The node association link appears between both nodes.



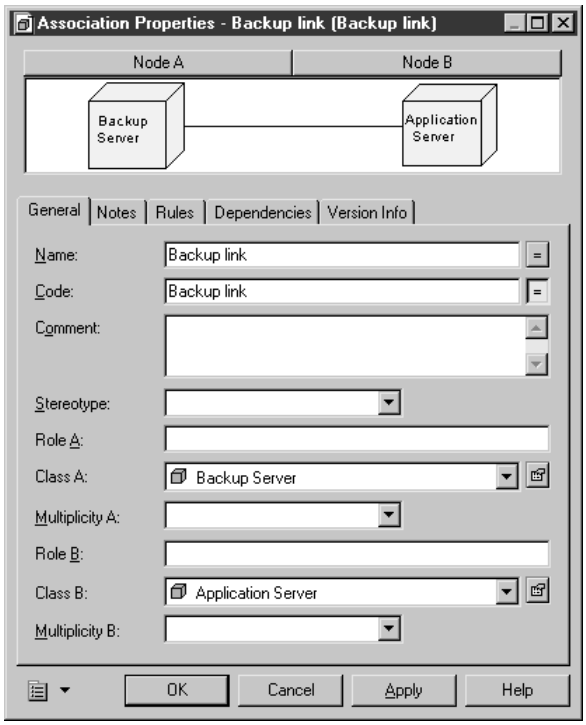
Dragging an association to a different node
You change the node at either end of a node association by clicking the node association to select it, pressing down CTRL, and dragging one of the attach points to a different node.

3 Click the pointer tool.
or
Click the right mouse button.

You release the Node Association tool.

4 Double-click the node association in the diagram to display the node association property sheet.

5 Type a name and a code.



You can open the property sheet of both nodes linked by the node association by clicking the Node A and Node B buttons located in the upper part of the property sheet.

6 Type a name for Role A and Role B if applicable.

7 Click OK.

Modifying node association properties

You can modify the node association properties using the following methods:

- From the property sheet of the node association
- From the list of Node Associations

For more information on the different ways to modify the node association properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying node association display preferences

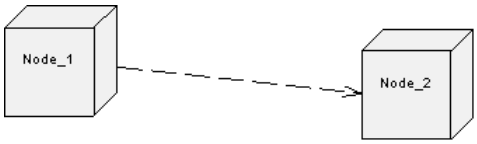
You can modify the following display preferences for node associations using the Tools→Display Preferences command:

Preference	Description
Show Multiplicity	Displays the cardinality of the relationship. You can choose between the actual number of instances (String) or a symbol at each end of the relationship (Symbol)
Show name	Displays the name of the node association
Show stereotype	Displays the stereotype of the node association
Show role names	Displays the name of the node association roles

Defining dependencies in a deployment diagram

A **dependency** is a relationship between two modeling elements, in which a change to one modeling element (the influent element) will affect the other modeling element (the dependent element).

The dependency relationship indicates that one node in a diagram uses the services or facilities of another node.



In a deployment diagram, a dependency can be created between nodes, and component instances.

Dependency properties in a deployment diagram

You can double-click any dependency symbol in a diagram to display its properties.

A dependency has the following properties:

Property	Description	Maximum length
Name	Name of the dependency	254
Code	Reference name of the dependency	254
Comment	Descriptive comment for the dependency	__
Influent	Node influencing the dependent node. Changes on the influent node affect the dependent node	254
Dependent	Node depending on the influent node. Changes on the dependent node will not affect the influent node	254
Stereotype	Extends the semantics of a dependency derived from existing dependencies but specific to your needs	254

Creating a dependency in a deployment diagram

You create a dependency between two nodes:

- From a diagram
- From the Browser

At creation, a dependency has a default name including a number, this number is assigned in the order of creation.

For more information on the different ways to create a dependency, see section Creating an object in chapter Managing objects in the *General Features Guide*.

Creating a dependency from a deployment diagram

❖ **To create a dependency from a deployment diagram:**

1 Click the Dependency tool.

2 Click inside the dependent node and while continuing to hold down the mouse button, drag the cursor to the influent node. Release the mouse button inside the influent node.

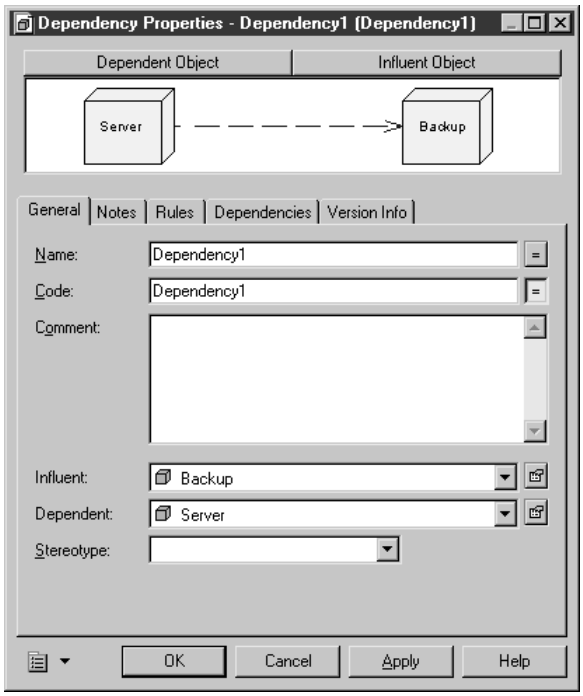
The link appears between the two nodes.

> **Dragging a dependency to a different node**
> You change the node at either end of a dependency by clicking the dependency to select it, pressing down CTRL, and dragging one of the attach points to a different node.

3 Click the Pointer tool.
or
Click the right mouse button to release the Dependency tool.

4 Double-click the dependency in the diagram to display the dependency property sheet.

5 Type a name and a code.



You can open the property sheet of the dependent and influent nodes by clicking the Dependent Object and Influent Object buttons located in the upper part of the dependency property sheet.

6 Click OK.

Modifying dependency properties in a deployment diagram

You can modify the dependency properties using the following methods:

- From the dependency property sheet
- From the list of dependencies

For more information on the different ways to modify the dependency properties, see section Modifying object properties in chapter Managing objects in the *General Features Guide*.

Modifying dependency display preferences

You can modify the following display preferences for dependencies using the Tools→Display Preferences command.

Preference	When selected
Show name	Displays the name of the dependency
Show stereotype	Displays the stereotype of the dependency
Show constraints	Displays the constraints (business rules) of the dependency

CHAPTER 11

Working with Platform Dependent Components

About this chapter

This chapter describes the platforms that support components of specific types and explains the characteristics of each of these components.

Contents

Topic	Page
Platform overview	406
Defining EJBs	408
Defining servlets	445
Defining JSPs	454
Defining ASP.NET	462

Platform overview

You can use two different platforms when working with components. They are also called Families within the context of the OOM and they contain different types of components.

The supported platforms are the following:

- Java (J2EE ™)
- .NET

Working with the Java family

The Java ™ 2 Platform, Enterprise Edition (**J2EE** ™) is a Java platform that defines the standard for developing multi-tier enterprise applications. J2EE simplifies enterprise applications by basing them on standardized, reusable modular components, it provides a complete set of services to those components, and handles many details of application behavior automatically without complex programming.

Java 2 Enterprise Edition supports the following technologies that you can use in PowerDesigner:

- Enterprise JavaBean (EJB)
- Java Servlets (Servlets)
- Java Server Page (JSP)

Working with the .NET family

The Microsoft .NET platform facilitates component-based development. It comprises the Common Language Runtime (CLR) that allows different languages to become a single language independent from the machine. The .NET family includes the Microsoft .NET Framework for building integrated, service-oriented applications to meet the needs of today's Internet businesses; applications that gather information from, and interact with a wide variety of sources, regardless of the platforms or languages in use. It also allows the creation of Web services as services that share information and combine functionality to provide tailored solutions.

The .NET family includes languages like C# and VB.NET that you can use in an OOM.

It supports the following technology:

- Microsoft Active Server Page (ASP.NET)

Defining EJBs

Sun Microsystem's definition of Enterprise JavaBeans ™ is:

"The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional and multi-user secure. These applications may be written once and then deployed on any server platform that supports the Enterprise JavaBeans specification".

EJBs are non-visual, remotely executable components that can be deployed on a server, they provide a component architecture framework for creating distributed n-tier middleware. They communicate with clients to make some services available to them.

Enterprise JavaBeans enable rapid and simplified development of distributed, transactional, secure, and portable Java applications. EJB servers reduce the complexity of developing middleware by providing automatic support for middleware services such as transactions, security and database connectivity.

For more information on EJB, a specification is available from Sun Web site, at http://java.sun.com

PowerDesigner supports the EJB 2.0 specification, with special emphasis on entity beans (both CMP and BMP). PowerDesigner's implementation of EJB support can take full advantage of the tight integration between PDM and OOM.

What do you need to work with EJBs?

You need Java 2 SDK Standard Edition (J2SE ™) 1.3 (final release), Java 2 SDK Enterprise Edition (J2EE ™) 1.3 (final release), a Java IDE or a text editor. You also need a J2EE application server supporting EJB 2.0.

It is also recommended that you set up the JAVA_HOME and J2EE_HOME system variables in your environment as follows:

In CLASSPATH:

```
%JAVA_HOME%\lib;%J2EE_HOME%\lib\j2ee.jar;%J2EE_HOME%\lib
```

In Path:

```
%JAVA_HOME%\bin;%J2EE_HOME%\bin
```

Using EJB types

You can define the following types of EJB components:

Type	Definition
Entity Beans	Designed to represent data in the database; they wrap data with business object semantics, read and update data automatically. Entity beans include: **CMP (Container Managed Persistence)** With CMP Entity Beans, persistence is handled by the component server (also known as Container) **BMP (Bean Managed Persistence)** With BMP Entity Beans, persistence management is left to the bean developer
Session Beans (Stateful and Stateless)	Encapsulate business logic and provide a single entry point for client users. A session bean will usually manage, and provide indirect access to several entity beans. Using this architecture, network traffic can be substantially reduced. There are Stateful and Stateless beans (see below)
Message Driven Beans	Anonymous beans that cannot be referenced by a given client, but rather respond to JMS asynchronous messages. Like Session Beans, they provide a way of encapsulating business logic on the server side

Entity Beans

Entity beans are used to represent underlying objects. The most common application for entity beans is their representation of data in a relational database. A simple entity bean can be defined to represent a database table where each instance of the bean represents a specific row. More complex entity beans can represent views of joined tables in a database. One instance represents a specific customer and all of that customer's orders and order items.

The code generated is different depending on the type of Entity Bean (Container Managed Persistence or Bean Managed Persistence).

Stateful and Stateless Session Beans

A session bean is an EJB in which each instance of a session bean is created through its home interface and is private to the client connection. The session bean instance cannot be easily shared with other clients, this allows the session bean to maintain the client's state. The relationship between the client and the session bean instance is one-to-one.

Stateful session beans maintain conversational state when used by a client. A conversational state is not written to a database, it is a state kept in memory while a client uses a session.

Stateless session beans do not maintain any conversational state. Each method is independent, and uses only data passed in its parameters.

Message Driven Beans

They are stateless, server side components with transactional behavior that process asynchronous messages delivered via the Java Message Service (JMS). Applications use asynchronous messaging to communicate by exchanging messages that leave senders independent from receivers.

EJB properties

You can open the EJB page in the component property sheet to display the following properties:

Property	Description
Remote home interface	Defines methods and operations used in a remote client view. Extends the javax.ejb.EJBHome interface
Remote interface	Provides the remote client view. Extends the javax.ejb.EJBObject interface
Local home interface	Defines methods and operations used locally in a local client view. Extends the javax.ejb.EJBLocal-Home interface
Local interface	Allows beans to be tightly coupled with their clients and to be directly accessed. Extends the javax.ejb.EJBLocalObject interface
Bean class	Class implementing the bean's business methods
Primary key class	Class providing a pointer into the database. It is linked to the Bean class. Only applicable to entity beans

Opening the EJB page
You can open the EJB page by right clicking the EJB component symbol in the diagram, and selecting EJB from the contextual menu.

Previewing the component code

You can preview the code of an EJB by clicking the Preview tab in the component property sheet. The Preview page contains tabbed pages that correspond to each interface and class of the EJB. You can see the relation between an EJB and its classes and interfaces from the Preview page without generating any file.

In the class and interface property sheets, the Preview page is an accelerator to show what will be generated.

In the model or package property sheet, the Preview page describes the EJB deployment descriptor file with the name of the generated EJB and its methods. It allows you to view what will be generated.

You can use the following tools and accelerators from the Preview toolbar:

Tool	Description	Accelerator
	Open Editor Contextual menu	SHIFT + F11
	Refresh	F5
	Show Generation Options	CTRL + W
	Select Generation Targets	CTRL + F6

Showing generation options

If you click the Show Generation Options tool, the Generation Options dialog box appears. You can change generation options from this dialog box and see the impact on the code.

Select Generation Targets

This tool is available when at least one extended model definition flagged for generation is linked to the model, and when it contains entries in the Generated Files category for the current type of object. When available, it displays the list of targets for the current object. If you add a generation target, the corresponding tab is added to the Preview page. If you deselect a generation target, the corresponding tab disappears from the Preview page.

For more information on setting an extended model definition for generation, see section Generating for an extended model definition, in chapter Extended Model Definitions Reference Guide, in the *Advanced User Documentation*.

Script bookmarks

In the Preview page, you can add and remove bookmarks at specific points in the code and then navigate forwards or backwards from bookmark to bookmark:

Accelerator	Description
CTRL + F2	Adds a new bookmark. A blue bookmark box appears. If you repeat this action from the same position, the bookmark is deleted and the blue marker disappears
F2	Jumps to bookmark
SHIFT + F2	Jumps to previous bookmark

Note that bookmarks are not printable and are lost if you use the Refresh, or Show Generation Options tools.

Creating an EJB with the wizard

You can create an EJB component with the **wizard** that will guide you through the creation of the component. It is only available if the language is Java.

The wizard is invoked from a **class diagram**. You can either create an EJB without selecting any class, or select a class first and start the wizard from the contextual menu of the class.

You can also create several EJBs of the same type by selecting several classes at the same time. The wizard will automatically create one EJB per class. The classes you have selected in the class diagram become the Bean classes. They are renamed to fit the naming conventions standard, and they are linked to their component.

If you have selected classes or interfaces before starting the wizard, they are automatically linked to the new EJB component.

When an interface or a class is already stereotyped, like <<EJBEntity>> for example, it is primarily used to be the interface or the class of the EJB.

For more information on stereotyped EJB interface or class, see section Defining interfaces and classes for EJBs.

The wizard for creation of an EJB lets you define the following parameters:

Property	Description
Name	Name of the EJB component
Code	Code of the EJB component
Component type	Entity Bean CMP, Entity Bean BMP, Message Driven Bean, Session Bean Stateful, or Session Bean Stateless For more information on the different types of EJB, see section Using EJB types
Bean class	Class implementing the bean's business methods
Remote interface	Extends the javax.ejb.EJBObject interface and provides the remote client view
Remote home interface	Defines methods and operations used in a remote client view. It extends the javax.ejb.EJBHome interface

Property	Description
Local interface	Extends the javax.ejb.EJBLocalObject interface, and allows beans to be tightly coupled with their clients and to be directly accessed
Local home interface	Defines methods and operations used locally in a local client view. It extends the javax.ejb.EJBLocal-Home interface
Primary key class	Class providing a pointer into the database. It is only applicable to entity beans
Transaction	Defines what transaction support is used for the component. The transaction is important for distribution across a network on a server. The transaction support value is displayed in the deployment descriptor. This information is given by the deployment descriptor to the server when generating the component
Create symbol	Creates a component symbol in the diagram specified beside the Create symbol In check box. If a component diagram already exists, you can select one from the dropdown listbox. You can also display the diagram properties by selecting the Properties tool
Create Class Diagram for component classifiers	Creates a class diagram with a symbol for each class and interface. If you have selected classes and interfaces before starting the wizard, they are used to create the component. This option allows you to display these classes and interfaces in a diagram

The Transaction support groupbox contains the following values, as per the Enterprise JavaBeans 2.0 specification from Sun's Microsystems, Inc.

Transaction value	Description
Not Supported	The component does not support transaction, it does not need any transaction and if there is one, it ignores it
Supports	The component is awaiting a transaction, it uses it
Required	If there is no transaction, one is created
Requires New	The component needs a new transaction at creation, the server must provide it with a new transaction
Mandatory	If there is no transaction, an exception is thrown
Never	There is no need for a transaction

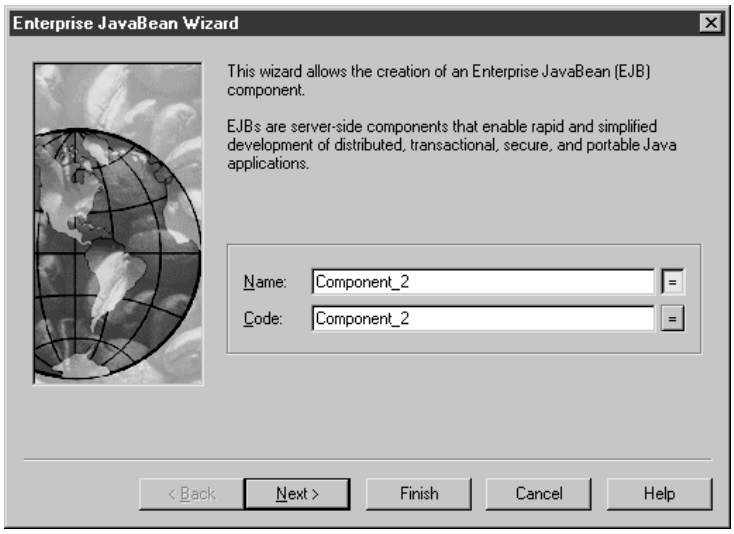
The EJB deployment descriptor supports transaction type for each method: you can then specify a transaction type for each method of EJB remote and local interface.

You can define the transaction type for each method using an extended attribute from the Profile/Operation/Extended Attributes/EJB folder of the Java object language. If the transaction type of the operation is not specified (it is empty), the default transaction type defined in the component is used instead.

❖ **To create an EJB with the wizard:**

1 Select Tools→Create Enterprise JavaBean from a class diagram.

The Enterprise JavaBean Wizard dialog box appears.



2 Type a name and code for the component and click Next.

If you have selected several classes, some of the pages are not displayed because the names are created by default according to the names of the selected classes.

3 Select the component type and click Next.

4 Select the Bean class name and click Next.

5 Select the remote interface and the remote home interface names and click Next.

6 Select the local interface and the local home interface names and click Next.

7 Select the primary key class name and click Next.

8 Select the transaction support and click Next.

9 At the end of the wizard, you have to define the creation of symbols and diagrams.

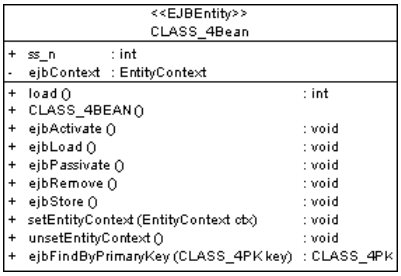
When you have finished using the wizard, the following actions are executed:

- An EJB component is created
- Classes and interfaces are associated with the component, and any missing classes or interfaces associated with the component are added
- Any diagrams associated with the component are created or updated
- Depending on the EJB type, the EJB primary key class, its interfaces and dependencies are automatically created and visible in the Browser. In addition to this, all dependencies between remote interfaces, local interfaces and the Bean class of the component are created
- The EJB created is named after the original class if you have selected a class before starting the wizard. Classes and interfaces are also prefixed after the original class name to preserve coherence

Adding a method to the Bean class

After creation of the Bean class, you may need to create a method that is missing at this stage of the process, such as an internal method.

The Bean class typically looks like the one below:


<<EJBEntity>>
CLASS_4Bean
+ ss_n : int
- ejbContext : EntityContext
+ load () : int
+ CLASS_4BEAN ()
+ ejbActivate () : void
+ ejbLoad () : void
+ ejbPassivate () : void
+ ejbRemove () : void
+ ejbStore () : void
+ setEntityContext (EntityContext ctx) : void
+ unsetEntityContext () : void
+ ejbFindByPrimaryKey (CLASS_4PK key) : CLASS_4PK

❖ To add an internal method to the Bean class:

1 Double-click the Bean class to display its property sheet.

2 Click the Operations tab to display the Operations page.

3 Click a blank line in the list.

An arrow appears at the beginning of the line.

4 Double-click the arrow at the beginning of the line.

A confirmation box asks you to commit the object creation.

5 Click Yes.

The operation property sheet appears.

6 Type a name and code for your operation.

7 Click the Implementation tab to display the Implementation page.

The Implementation page opens to the Body tabbed page.

8 Add the method code in this page.

9 Click OK.

You return to the class property sheet.

10 Click the Preview tab to display the Preview page.

You can now validate the to-be-generated Java code for the Bean class.

11 Click OK.

12 Double-click the component diagram node containing your EJB component from the Browser.

The diagram displays the EJB component.

Understanding EJB support in an OOM

PowerDesigner simplifies the development process by transparently handling EJB concepts and enforcing the EJB programming contract through the following levels of support:

- Automatic initialization: occurs whenever a class or interface is attached to an EJB
- Synchronization: occurs between interfaces and classes already attached to a component. It may involve adding a **linked method**

  For more information on the linked method, see section Understanding EJB synchronization.
- Model checks: the Check Model feature validates a given model and complements synchronization by offering auto-fixes. You can check your model at any time using the Check Model feature from the Tools menu.

  For more information on the Check Model feature, see section Checking an OOM, in chapter Working with Object-Oriented Models.

- Template based code generation: supports the programming contract defined in the specification and allows you to customize generated code. EJB specific templates can be found in the Java object language in the Profile/Component/Templates category, available from Language→Edit Current Object Language.

For more information, see sections Responsibilities of the Bean Provider, in all chapters related to Bean component contract in the Enterprise JavaBeans 2.0 specification from Sun's Microsystems, Inc.

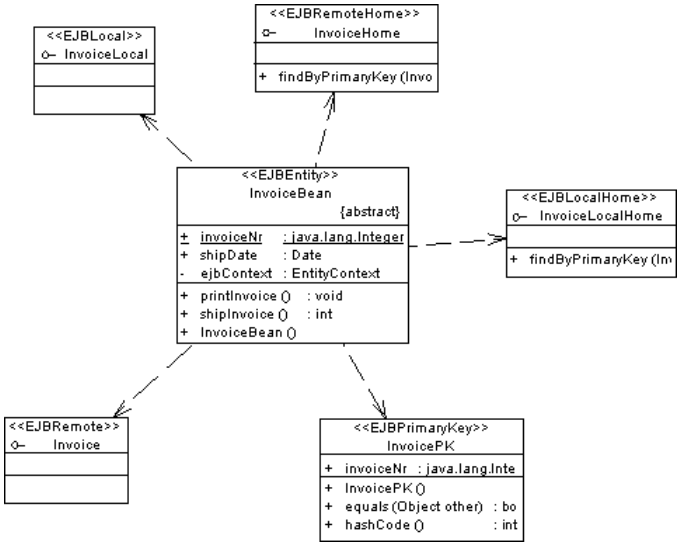
At a higher level, PowerDesigner supports different approaches to assist you in the component development process including:

- Forward engineering: from an OOM to a PDM. It provides the ability to create and reverse EJBs in a OOM, generate the corresponding PDM, O/R Mapping and generate code
- Reverse engineering: from a PDM (database) to an OOM. It provides the ability to create and reverse tables in a PDM, generate corresponding classes, create EJB from given classes and generate code

Understanding EJB initialization

When creating an EJB, the role of the initialization process is to initialize classes and interfaces with respect to the EJB programming contract. It instantiates the classes and interfaces together with their methods.

Example



Understanding EJB synchronization

Synchronization maintains the coherence of the whole model whenever a change is applied. It occurs progressively as the model is modified.

PowerDesigner successively performs several actions to complete synchronization at the model level.

Example

Double-click the Bean class of a component, click the Operations tab, click the Add button at the bottom of the Operations page, and select EJB Create method (local).

When you add the create<METHOD> from the Bean class, some actions are performed onto the local home interface: PowerDesigner sets the stereotype of the create<METHOD> to <<EJBCreateMethod>> stereotype. This stereotype is then added to the local home interface.

It also sets the return type of the create<METHOD> to the local interface by filling in the Return Type dropdown listbox with the name of the local interface.

Other actions are successively invoked until synchronization is complete.

How synchronization occurs?

The following table gives a description of some events that happen whenever a modification occurs on a component:

Event	Description
Initialize()	Defined on all objects, corresponds to the initialization of the object
Uninitialize()	Triggers the uninitialization of the object
SetType(type)	Occurs when the type of the component changes (see the Type dropdown listbox in the component property sheet)
SetCode(code)	Occurs when the code of the component changes. Code: new code
Attach(classifier)	Occurs when an interface or class gets attached to a component (from the EJB page of the component property sheet). Classifier: classifier being attached
Detach(classifier)	Occurs when an interface or a class is detached from a component (from the EJB page of the component property sheet). Classifier: classifier being detached

As an example, for some of the above events, actions that are performed involve the following synchronization:

Event	What synchronization occurs....
SetType( )	Example with an EJB Entity Bean CMP: For each attached classifier: Uninitialize(), then creates or attaches if they already exist, and initializes or synchronizes remote home interface, remote interface, primary key class and Bean class
SetCode( )	For each attached classifier: Rename() if the code of the classifier follows naming conventions
Attach( )	Example for HomeInterface: HomeInterface: Initialize() and BeanClass: Synchronize() with home interface, then creates a dependency link between Bean class and home interface
	Example for BeanClass: BeanClass: Initialize(), then Synchronize() with home interface, then Synchronize() with local interface, PrimaryKeyClass: Synchronize() with Bean class, then creates a dependency links towards other classifiers

Linked method

In general, all methods that appear grayed from the list of operations in EJB classes are linked methods. A linked method has its name or code, return type and parameters synchronized with the method it is tied to. It must respect the contract with its related EJB, so when changes occur on the EJB, its linked methods are updated.

Understanding EJB code generation

The code generation mechanism is only described for Entity beans CMP in this section since the code generated is handled and stored in the Container. With Entity beans BMP, session beans and message-driven beans, the code generated is highly dependent on the bean developer's input, it would thus not be possible to introduce each possible configuration here.

The process of code generation is explained below in order for you to understand the mechanism:

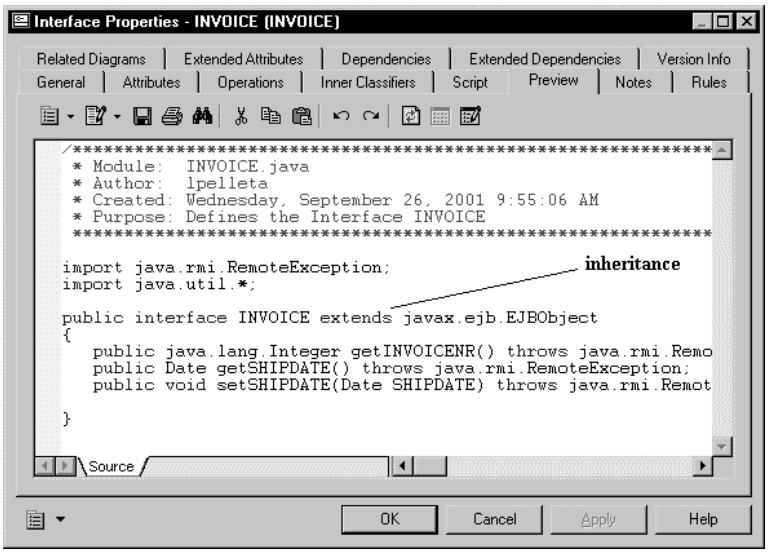
The generation of EJB class and interface into code is performed like if they were Java classes, except that they are dependent on an EJB and thus contain more information to be processed during generation.

The code generation process performs the following tasks:

1 It creates the following inheritance links:

The local home interface inherits javax.ejb.EJBLocalHome
The local interface inherits javax.ejb.EJBLocalObject
The remote home interface inherits javax.ejb.EJBHome
The remote interface inherits javax.ejb.EJBObject

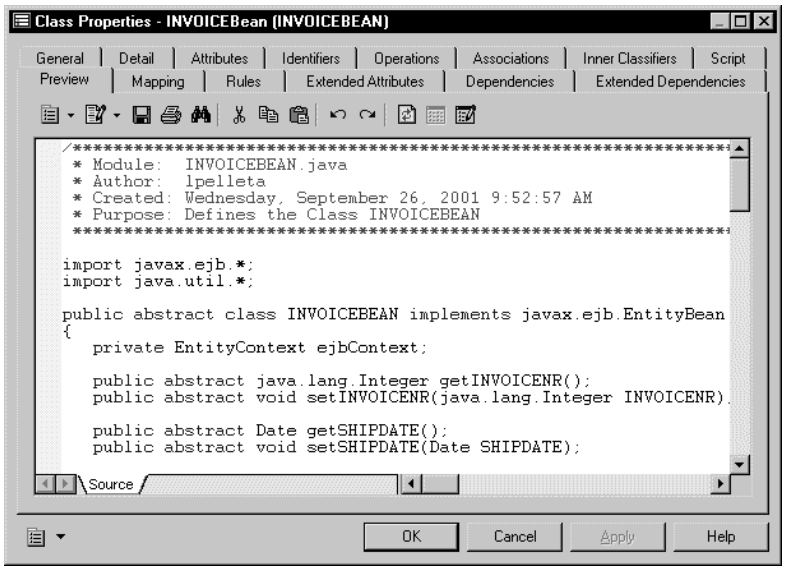
Here is an example from the Preview page of the remote interface:



2 It creates the following realization links:

The Primary key class implements java.io.Serializable
The Bean class implements javax.ejb.EntityBean

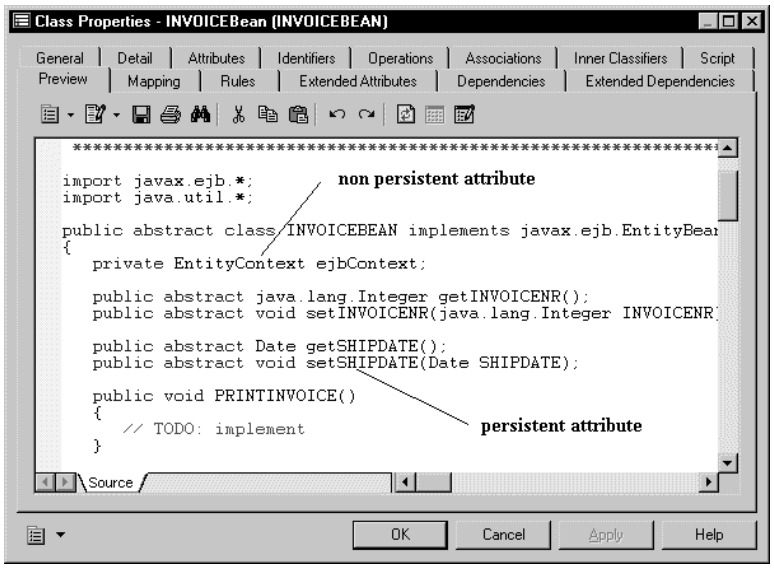
Here is an example from the Preview page of the Bean class:



3 It transforms attributes flagged as persistent into getter and setter methods (cmp-fields and cmr-fields are generated as getters and setters).

cmp-fields are attributes flagged as Persistent in the property sheet. cmr-fields are attributes that are migrated from associations.

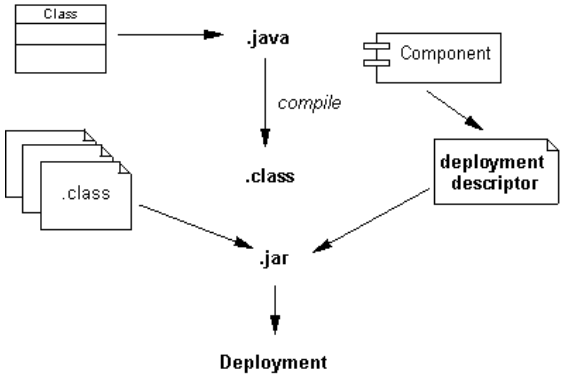
Here is an example from the Preview page of the Bean class:



Using the EJB deployment descriptor

An EJB deployment descriptor describes the structure and properties of one or more EJBs in an XML file format. It is used for deploying the EJB in the application server. It declares the properties of EJBs, the relationships and the dependencies between EJBs. One deployment descriptor is automatically generated per package or model, it describes all EJBs defined in the package.

The deployment descriptor's role, as part of the whole process is shown in the following figure:



The EJB deployment descriptor and the compiled Java classes of the EJBs should be packaged in a JAR file.

The EJB deployment tool of the application server imports the Java classes from the JAR file and configures the EJBs in the application server based on the description of EJBs contained in the EJB deployment descriptor.

You can see the deployment descriptor from the Preview page of the package or model property sheet.

For more information on the EJB deployment descriptor, see section Generating EJB deployment descriptor.

You can customize the EJB deployment descriptor by modifying the templates in the Java object language.

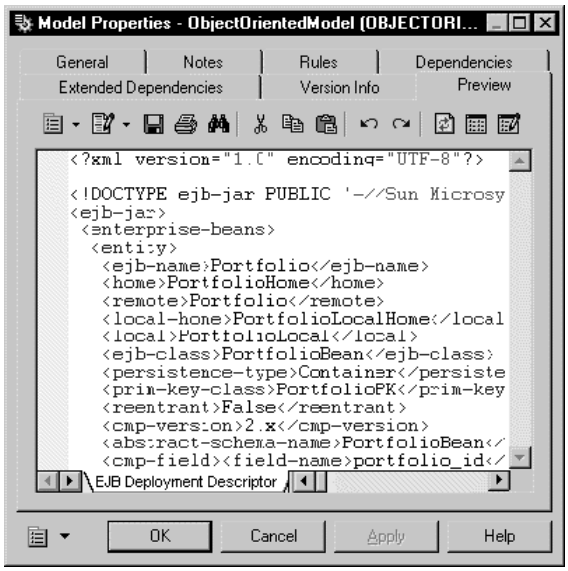
For more information on customizing the object language, see section Understanding the object language editor, in chapter Object Languages Reference Guide in the *Advanced User Documentation*.

❖ **To view the EJB deployment descriptor:**

1 Select Model→Model Properties
or
Right-click the diagram background and select Properties from the contextual menu.

The model property sheet appears.

2 Click the Preview tab.

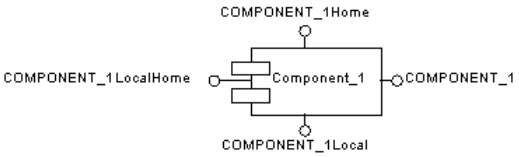


The Preview tab displays the deployment descriptor content.

Defining interfaces and classes for EJBs

An EJB comprises a number of specific interfaces and implementation classes. Interfaces of an EJB are always exposed, you define a public interface and expose it. You can attach an interface or class to only one EJB at a time.

EJB component interfaces are shown as circles linked to the EJB component side by an horizontal or a vertical line:



Interfaces provide a remote, or a local view as shown in the table below:

View	Interface
Remote view	Remote home interface Remote interface
Local view	Local home interface Local interface

The symbol of an interface can be moved, and the link from the EJB component to the interface can be extended. When an interface realizes a EJB component, the interface is necessarily visible.

Displaying the interface properties
You can double-click the circle, symbol of an interface to display the interface property sheet.

Classes have no symbol in the component diagram, unlike interfaces. However, the relationship between the class and the EJB component is visible from the Classes page of the EJB component property sheet, and from the Components tabbed page in the Dependencies page of the class property sheet.

The following table displays the stereotypes used to automatically identify EJB interfaces and classes:

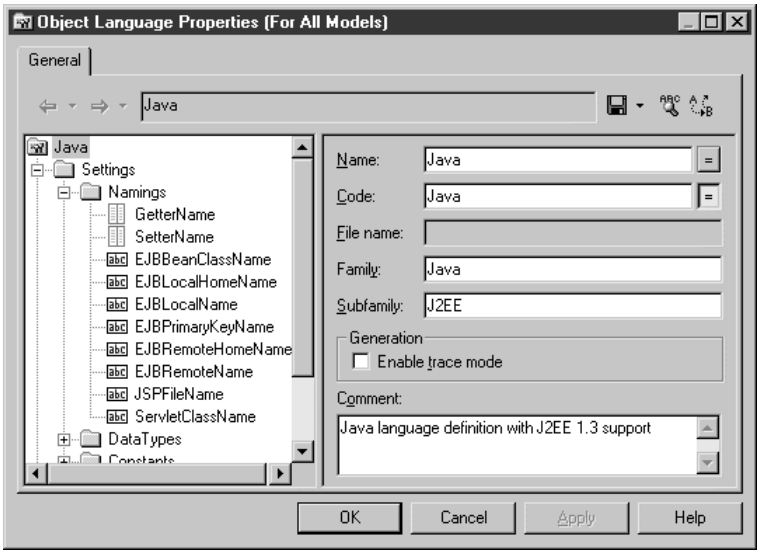
Stereotype	Describes
<<EJBRemoteHome>>	The remote home interface
<<EJBRemote>>	The remote interface
<<EJBLocalHome>>	The local home interface
<<EJBLocal>>	The local interface
<<EJBEntity>>	The bean class of an entity bean
<<EJBSession>>	The bean class of a session bean

Stereotype	Describes
<<EJBMessageDriven>>	The bean class of a message driven bean
<<EJBPrimaryKey>>	The primary key class of an entity bean

Naming conventions

A **naming convention** is a common model option that you can define in the Model Options dialog box. It is useful to set rules in order to enforce predefined name formats for all the objects within a model. You can specify naming convention for related interfaces and classes in your object language in the Settings→Namings category, available from Language→Edit Current Object Language.

For more information on naming conventions, see section Defining naming conventions, in chapter Managing Models in the *General Features Guide*.



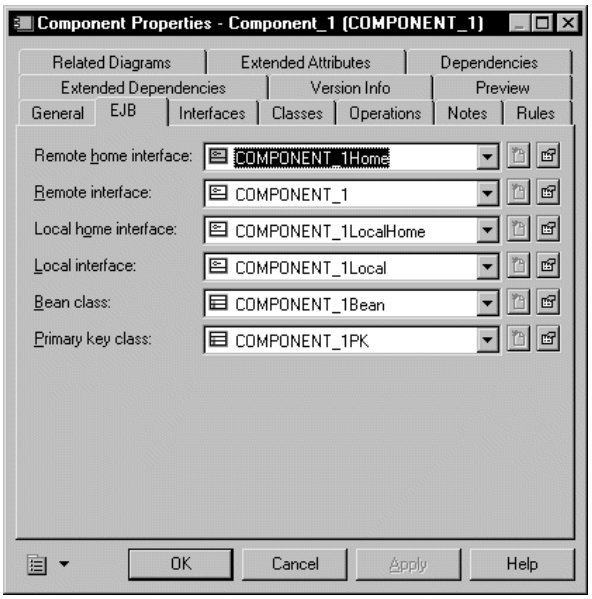
Template names are instantiated with respect to the corresponding component and assigned to the newly created objects. If an unattached interface or class, matching a given name and classifier type already exists in the model, it is automatically attached to the EJB.

❖ To define an interface or a class for an EJB:

1 Right-click the EJB component in the diagram and select EJB from the contextual menu.

The component property sheet opens to the EJB page.

Interfaces and classes are created and attached to the EJB.



You can use the Create tool beside the interface or the class name to recreate an interface or a class if it is set to <None>.

2 Click the Properties button beside the interface or the class name box that you want to define.

The interface or the class property sheet appears.

3 Select properties as required.

The interfaces and classes definitions are added to the current EJB component definition.

Defining operations for EJBs

If you want to create operations for an EJB, you can do so from the property sheet of the Bean class or EJB interfaces.

You cannot create an operation from the Operations page of the component property sheet as this page is only used to view the operations of the EJB component. You view operations of an EJB from the Operations page in the component property sheet.

Viewing EJB operations

The displayed EJB operations are operations of its interface(s).

❖ **To view EJB operations:**

1 Double-click the component in the diagram to display the component property sheet.

2 Click the Operations tab to display the Operations page.

3 Select one interface or <All interfaces> from the dropdown listbox.

The operations of the interface appear in the list. You can double-click an arrow at the beginning of a line to display an operation property sheet.

Creating operations from a Bean class or EJB interface

You can create the following operations for an EJB from the Bean class or an interface property sheet. You can create all these operations from the Bean class for example, since you have access to all interfaces from this class.

- EJB Create Method (local)
- EJB Create Method (remote)
- EJB Finder Method (local)
- EJB Finder Method (remote)
- EJB Select Method

For example, select the EJB Finder Method (remote) to add the find () method to the remote home interface.

Stereotypes

The following standard stereotypes, as defined for EJBs, are assigned to these operations:

<<EJBCreateMethod>>
<<EJBFinderMethod>>
<<EJBSelectMethod>>

CMP entity beans

You can create the following operations for CMP Entity Beans only:

- EJB Create(...) Method (local)
- EJB Create(...) Method (remote)

When you create an EJB Create(...) Method (local), the method is created in the local home interface with all the persistent attributes as parameters. When you create an EJB Create(...) Method (remote), the method is created in the remote home interface with all the persistent attributes as parameters.

Moreover, all linked methods ejbCreate(...) and ejbPostCreate(...) are created automatically in the Bean class with all the persistent attributes as parameters. Parameters are synchronized whenever a change is applied.

Modifying an interface method
If you need to modify the methods of an interface, you can do so from the Operations page of the interface property sheet.

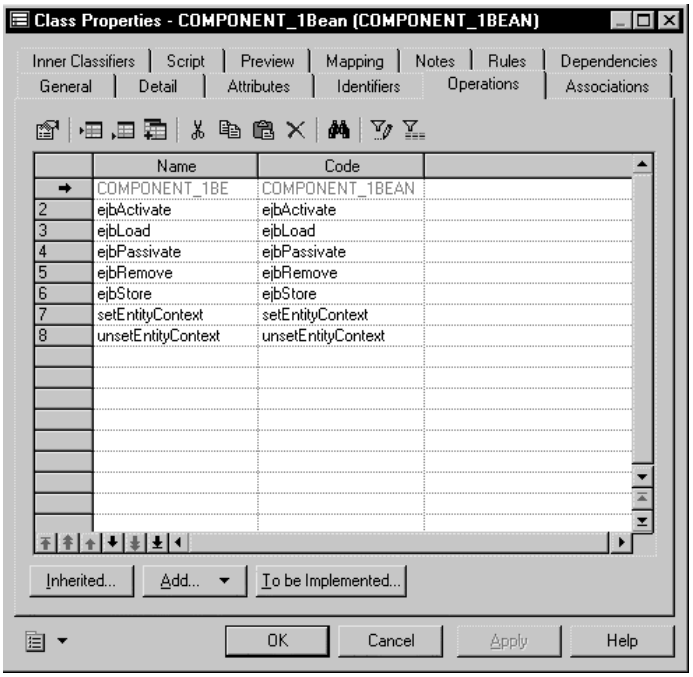
❖ **To create operations from a Bean class or EJB interface:**

1 Double-click the Bean class to display the class property sheet (or an EJB interface to display the interface property sheet).

2 Click the Operations tab to display the Operations page.

3 Click the Add button at the bottom of the page.

4 Select the required EJB operation from the dropdown listbox.



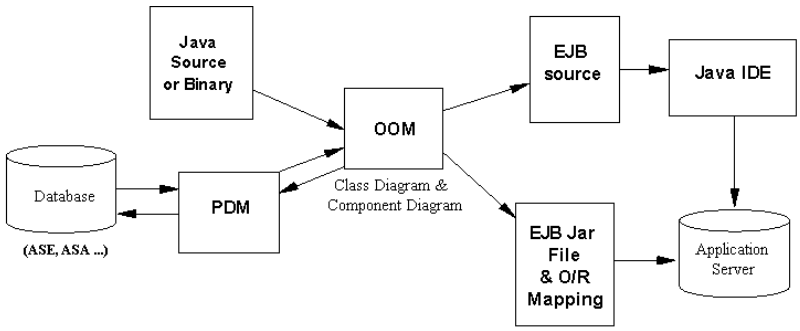
The requested operation is created at the end of the list of operations for the class or the interface.

5 Click OK.

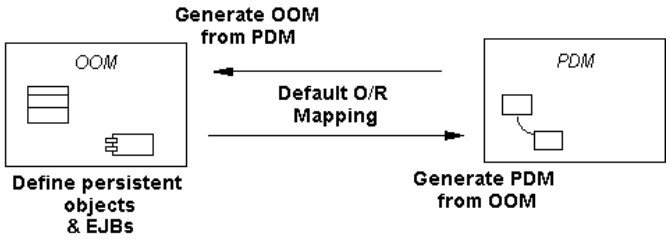
Generating EJBs

The EJB generation process allows you to generate EJB source code that is compliant with J2EE 1.3 (EJB 2.0). The code generator (Java, EJB, XML, etc…) is based on templates and macros. You can customize the generated code by editing the Java object language from Tools→Resources→Object Languages.

The following picture illustrates the overall EJB development process.



The following picture focuses on the PowerDesigner part, and highlights the role of O/R Mapping generation. The O/R mapping can be created when generating a PDM from an OOM or generating an OOM from a PDM.



For more information on object mapping, see sections Automatic creation of O/R mapping, and User-defined O/R mapping, in chapter Working with Object-Oriented Models.

When generating an EJB, a typical generation process could be to:

- Generate EJB source code, deployment descriptor, and O/R Mapping descriptor
- Compile Java file
- Create a Jar file

- Deploy into an application server

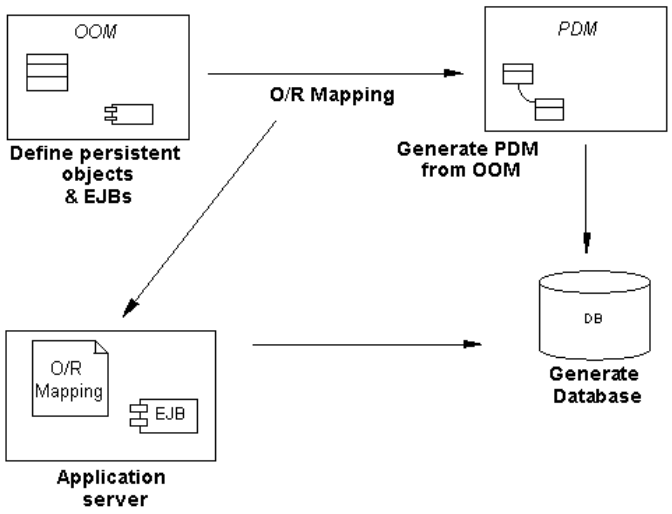
What kind of generation to use?

You may face two situations when generating EJB source code:

- You have no database
- You already have a database

You want to generate EJB source code when creating a database. If there is no database, the development process is the following:

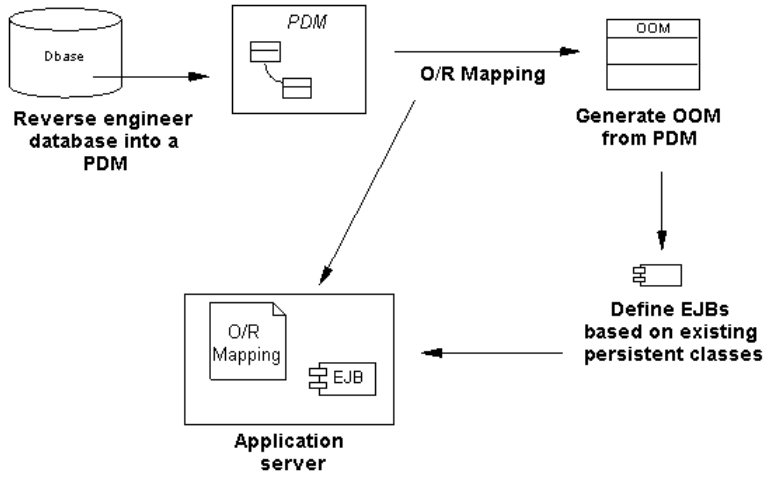
- Define persistent objects in the class diagram, and define EJBs in the component diagram
- Generate the Physical Data Model, with creation of an O/R Mapping during generation
- Create the database
- Generate EJB source code and deployment descriptor for deployment in the application server



You want to generate EJB source code for an existing database. If the database already exists, you may want to wrap the existing database into EJBs and use EJBs to access the database.

The development process is the following:

- Reverse engineer the PDM into an OOM, and retrieve the O/R mapping generated during reverse
- Define EJBs in the OOM based on persistent classes
- Generate the EJB source code and deployment descriptor for deployment in the application server



You can also use generation of EJB source code when managing persistence of EJBs with an existing database. For example, it may be necessary to manage the persistence of an already defined EJB with an existing database. Since you cannot change the definition of EJB nor the database schema, you need a manual object to relational mapping in this case.

For more information on object mapping, see sections Automatic creation of O/R mapping, and User-defined O/R mapping, in chapter Working with Object-Oriented Models.

Understanding EJB source and persistence

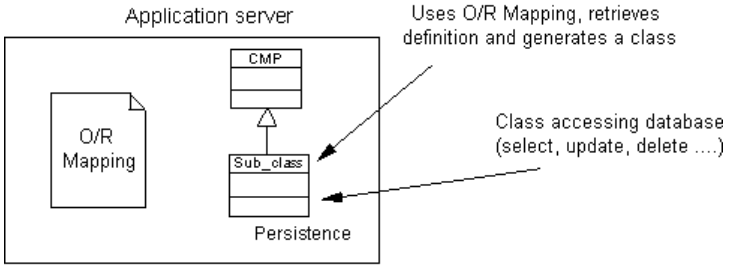
You generate persistence management methods based on the object language. Depending if the EJB is of a CMP or BMP type, the deployment descriptor file appears different:

- A CMP involves the application server. It includes the EJB and the O/R mapping descriptor (.XML). The server retrieves both EJB and O/R mapping descriptor to generate the code

If the application server does not support an O/R Mapping descriptor, the mapping must be done manually. If the O/R mapping descriptor of your application server is not supported yet, you can create your own by creating a new extended model definition.

For more information on creating an extended model definition, see section Creating an extended model definition, in chapter Extended Model Definitions Reference Guide, in the *Advanced User Documentation*.

- A BMP involves a manual process. It includes the EJB source, without any O/R mapping descriptor (O/R mapping is not necessary). The BMP developer should implement the persistence management him/herself by implementing the ejbStore(), and ejbLoad() methods, the application server only supports its functions. An implementation class inherits from the BMP Bean class, handles persistence data and communicates with the database
- You can also define an EJB as CMP, then generate it as BMP when generating the code. The code generator generates an implementation class (sub-class) for the Bean class that contains its methods, and uses an O/R mapping and a persistent template to implement the persistence



For more information on defining O/R mapping, see section Defining O/R mapping, in chapter Working with Object-Oriented Models.

You can use different methods to generate an EJB CMP into an EJB BMP. You can either copy the Java object language delivered in PowerDesigner as a reference to a new object language, and describe how to generate implementation classes of the EJB CMP in your own object language, or you can create an extended model definition that includes these implementation classes.

You could also write a VB script to convert the EJB CMP into an EJB BMP. To do this, you must generate the EJB as CMP, then launch the VB script that will go through all objects of the model and generate an implementation class for each identified class.

Generating EJB source code

When generating an EJB, classes and interfaces of the EJB are directly selected. The generation retrieves the classes used by the component, as well as the interfaces associated with the classes.

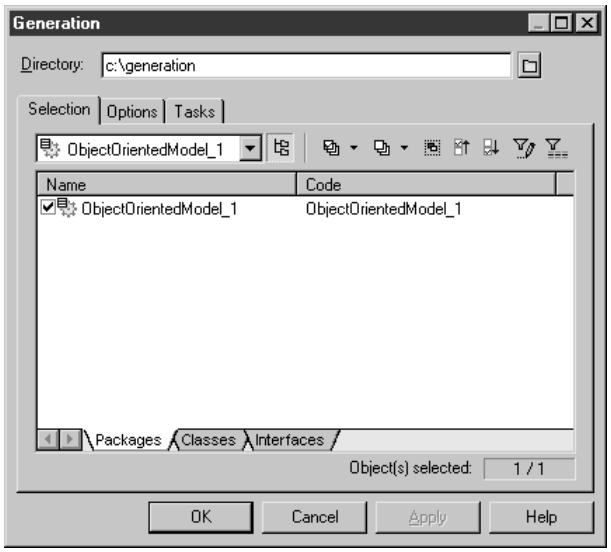
❖ **To generate EJB source code:**

1 Select Language→Generate Java Code to display the Generation dialog box.

2 Select or browse to a directory that will contain the generated Java files.

3 Click the Selection tab to display the Selection page.

4 Select the objects in the different tabbed pages.

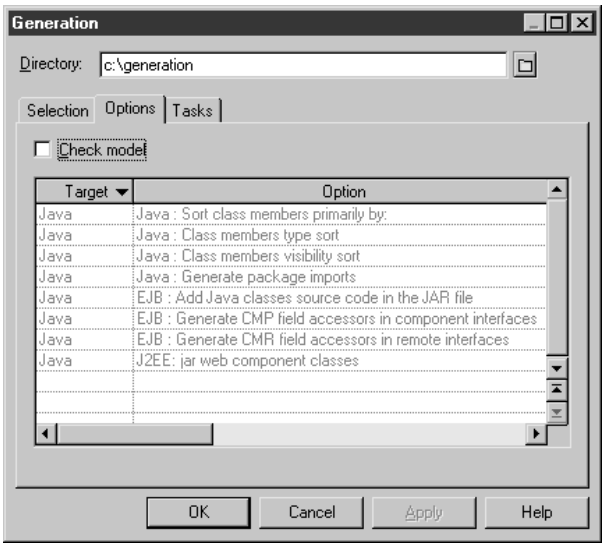


For more information on the selection page, see section Selecting objects to generate, in chapter Generating for a Language.

5 Click Apply.

6 Click the Options tab to display the Options page.

7 Specify your generation options in the Options page.

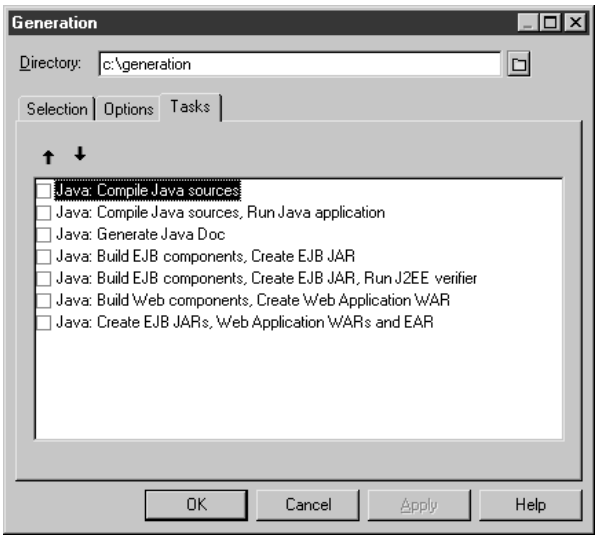

Generation
Directory: c:\generation
Selection Options Tasks
Check model
Target
Option
Java Java : Sort class members primarily by:
Java Java : Class members type sort
Java Java : Class members visibility sort
Java Java : Generate package imports
Java EJB : Add Java classes source code in the JAR file
Java EJB : Generate CMP field accessors in component interfaces
Java EJB : Generate CMR field accessors in remote interfaces
Java J2EE: jar web component classes
OK
Cancel
Apply
Help

For more information on the generation options, see section Defining generation options, in chapter Generating for a Language.

8 Click Apply.

9 Click the Tasks tab to display the Tasks page.

10 Select the commands you want to perform during generation in the Tasks page.


Generation
Directory: c:\generation
Selection Options Tasks
Java: Compile Java sources
Java: Compile Java sources, Run Java application
Java: Generate Java Doc
Java: Build EJB components, Create EJB JAR
Java: Build EJB components, Create EJB JAR, Run J2EE verifier
Java: Build Web components, Create Web Application WAR
Java: Create EJB JARs, Web Application WARs and EAR
OK
Cancel
Apply
Help

For more information on the generation tasks, see section Defining generation tasks, in chapter Generating for a Language.

You must beforehand set your environment variables from General Options→Variables in order to activate them in this page.

For more information on how to set these variables, see section Defining environment variables, in chapter Using the PowerDesigner interface, in the *General Features Guide*.

11 Click OK.

A progress box appears, followed by a Result list. You can use the Edit button in the Result list to edit the generated files individually.

12 Click Close.

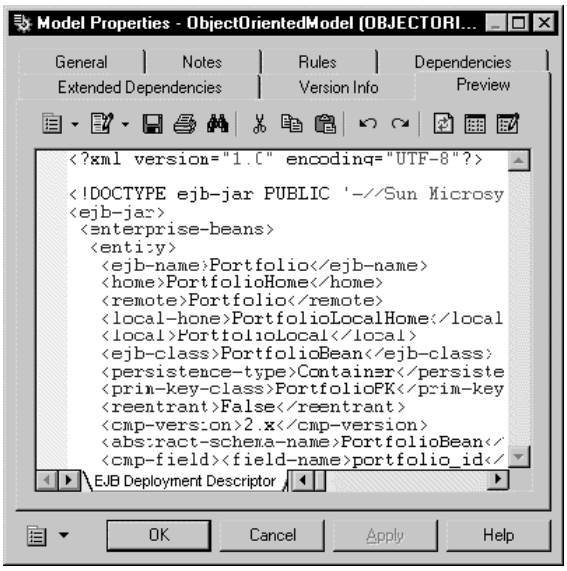
The ejb-jar.XML is created in the META-INF directory and all files are generated in the generation directory.

Generating EJB deployment descriptor

The EJB deployment descriptor is an XML file, called ejb-jar.XML, generated in the META-INF directory. It is independent from the application server.

For more information on generating the EJB deployment descriptor, see section Generating EJB source code.

The EJB deployment descriptor is available from the Preview page of the package, or the model property sheet.



Generating O/R mapping for an application server

Object to relational mapping allows you to generate objects from an Object-Oriented Model into a Physical Data Model. It is the process of having classes, attributes and operations mapped to relational tables, columns and queries by using DBMS compliant codes for OOM objects generated into the PDM. The purpose is to set up object storage in a relational database.

Object to relational mapping is used in the current context to generate an O/R mapping descriptor for EJBs CMP or EJB BMP implementation in an application server. You must define an O/R mapping description file format to be used for EJB implementation in your application server.

For more information on how to define the O/R mapping description file, see section Defining O/R mapping, in chapter Working with Object-Oriented Models.

Delivered samples

It is necessary to add a specific extended model definition tied to a server so that the server understands the format of the O/R mapping description file. You may import the extended model definition at any time, even before the creation of your EJBs. You can use the following extended model definition samples delivered in the Resource Files folder of PowerDesigner:

- Weblogic6.XEM

- EAServer4.XEM
- CocoBase4.XEM

For more information on how to import extended model definitions, see section Choosing a target application server.

It is recommended to read the appropriate technical document located in the Printable Docs directory of your installation, prior to using the extended model definition file.

Based on these samples, you can create your own extended model definition tied to the application server of your choice. Verify that the version of EJB supported in your application server is EJB 2.0.

Technical documents

You can find technical documents on other model extensions that are available for PowerDesigner. They are located in the Printable Docs directory of your installation.

Choosing a target application server

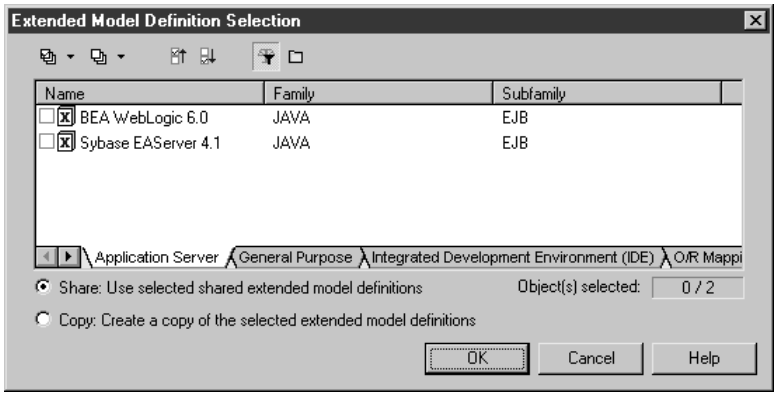
You can generate the specific descriptors of an application server, or choose an O/R mapping tool as follows:

❖ **To choose a target application server:**

1 Select Model→Extended Model Definitions to display the List of Extended Model Definitions.

2 Click the Import an Extended Model Definition tool.

The Extended Model Definition Selection page appears.



3 Select the required extended model definition.

4 Select either Share or Copy.

Share allows you to share the extended model definition file stored in the installation directory. Copy allows you to create a copy of the extended model definition file in the model.

5 Click OK.

The list of Extended Model Definitions displays the selected file.

6 Click OK.

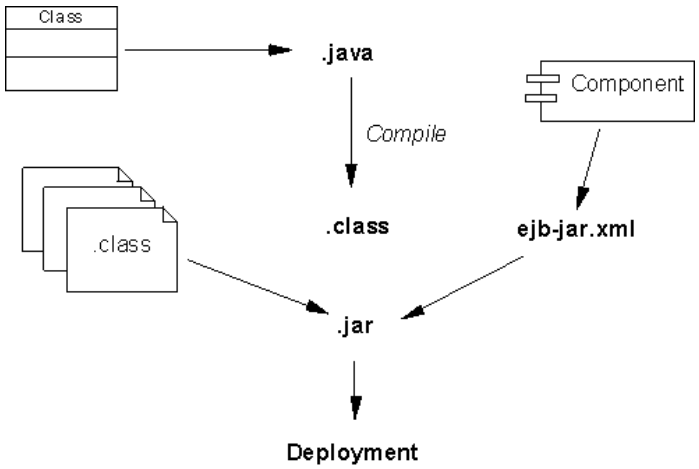
The new node for the extended model definition appears in the Browser.

Generating .JAR files

In order to package the EJB, the bean classes, interfaces and the deployment descriptor are placed into a .JAR file.

You can generate .JAR files using the appropriate generation task in the Tasks page of the Generation dialog box. It compiles .JAVA files using a compiler and creates a .JAR file including the compiled Java classes, then it completes the .JAR file with the deployment descriptor and icons.

This process is common to all EJB components.



> ***Caution***
> *You must set the values of the commands used during generation from General Options→Variables in order to enable them. For example, you can set the javac.exe and jar.exe executables at this location.*

For more information on how to set these variables, see section Defining environment variables, in chapter Using the PowerDesigner interface in the *General Features Guide*.

Reverse engineering EJBs

You can reverse engineer EJB into an OOM for the Java language only. The reverse engineering feature processes .JAR files, Class or Java directories and reverses the deployment descriptor.

Reverse engineering EJB options

You start reverse engineering EJB from Language→Reverse Engineer Java. Select one of the following Java formats from the Selection page in order to select the Reverse Engineer Deployment Descriptor check box from the Options page:

- Java directories
- Class directories
- Archive (.JAR file)

For more information on the Java formats, see section Reverse engineering Java formats in chapter Reverse engineering.

The reverse engineering feature looks for a deployment descriptor file located inside the .JAR, or the class or Java directories and creates the appropriate components. The deployment descriptor describes the components of the .JAR file, thus, for each described component, the reverse feature creates a component in the package specified by its bean class. If the Bean class is missing, the components are not reversed.

For more information on the deployment descriptor, see section Using the EJB deployment descriptor.

Reverse engineering EJB deployment descriptor

EJB components are defined inside the <enterprise-beans> element. Associations between bean classes are described inside the <relationships> element, in addition, EJB types are defined inside the following elements:

- Each Entity Bean is defined inside an <entity> element
- Each Session Bean is defined inside a <session> element
- Each Message Driven Bean is defined inside a <message-driven> element

<entity> example

The Entity Bean EJB type is further described as an example in this section.

Attributes

The following table maps elements to the following properties:

Element	Property
<description>	Component comment
<display-name>	Component name
<ejb-name>	Component code
<home>	Remote Home Interface code
<remote>	Remote Interface code
<local-home	Local Home Interface code
<local>	Local Interface code
<ejb-class>	Bean class code
<persistence-type>	'Container' for CMP and 'Bean' for BMP
<prim-key-class>	Primary key class code
<reentrant>	Component EXA (reentrant = true/false)
<cmp-version>	Component EXA (cmpVersion)
<abstract-schema-name>	Component EXA (abstractSchemaName)

<cmp-field>

For each persistent attribute, a cmp-field is generated. The <cmp-field-name> contains the attribute code.

<primkey-field>

If the primary key consists in only one attribute whose data type is one of the java.lang, the primary key class may be empty. In this case, a <primkey-field> containing the primary key attribute code is generated.

<ejb-ref>

The <ejb-ref> is used for declaration of a reference to an enterprise bean's home. It is generated when a dependency with <<ejb-ref>> stereotype is linked to the referenced bean in a different package.

ejb-ref element	Result in the component diagram
<ejb-ref-name>	EJB/{Referenced EJB code}
<ejb-ref-type>	Entity or Session bean
<home>	Referenced EJB Home Interface code
<remote>	Referenced EJB Remote Interface code
<ejb-link>	{Qualified Referenced EJB Jar name}#{Referenced EJB code}

<ejb-local-ref>

The <ejb-local-ref> is used for declaration of a reference to an enterprise bean's local home. It is generated when a dependency with <<ejb-ref>> stereotype is linked to the referenced bean in a different package.

ejb-ref element	Result in the component diagram
<ejb-ref-name>	EJB/{Referenced EJB code}
<ejb-ref-type>	Entity or Session bean
<local-home>	Referenced EJB Local Home Interface code
<local>	Referenced EJB Local Interface code
<ejb-link>	Referenced EJB code

<relationships> example

Associations between bean classes are further described as an example in this section.

Inside the <relationships> element, each <ejb-relation> defines one association between two bean classes.

Attributes

The following table maps elements to the following properties:

Element	Property
<ejb-relation-name>	Association code

<ejb-relationship-role>

Each association end is defined in an <ejb-relationship-role> element.

ejb-ref element	Result in the component diagram
<ejb-relationship-role-name>	Role name
<multiplicity>	Role multiplicity (One or Many)
<cascade-delete>	Association EXA (roleACMRCascadeDelete or roleBCMRCascadeDelete)

<relationship-role-source>

This element specifies the EJB whose bean class is associated with the current relationship role.

ejb-ref element	Result in the component diagram
<ejb-name>	EJB code

<cmr-field>

In a navigable role, the migrated attribute is generated as a <cmr-field>.

ejb-ref element	Result in the component diagram
<cmr-field-name>	Migrated attribute code

Reverse engineering EJB components

You reverse engineer EJB components located in .JAR files using the following method (the same method applies to Class and Java directories):

❖ **To reverse engineer EJB components:**

1 Select Language→Reverse Engineer Java.

The Reverse Java dialog box appears.

2 Select Archive from the Reverse dropdown listbox in the Selection page.

3 Click the Add button.

A standard Open dialog box appears.

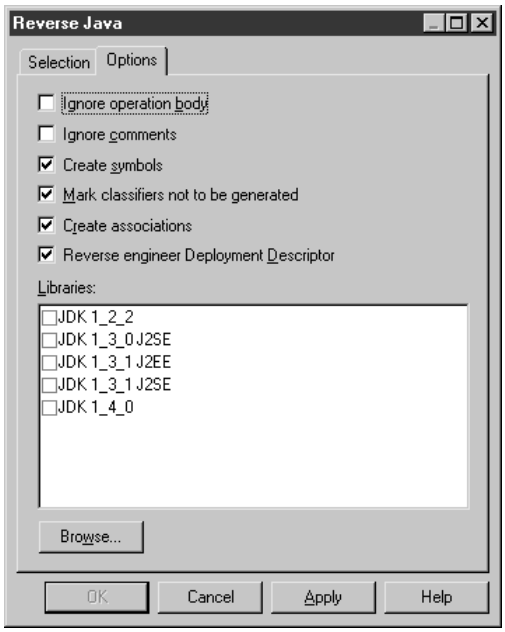
> **Using the Remove button**
> You can click the Remove button to remove a selection from the list.

4 Select the items you want to reverse and click Open.

The Reverse Java dialog box displays the items you selected.

5 Click the Options tab to display the options page.

6 Select the Reverse Engineer Deployment Descriptor check box.



7 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

8 Click OK.

The Reverse page of the Output window displays the changes that occurred during reverse and the diagram window displays the updated model.

Defining servlets

Servlets are programs that help in building applications that generate dynamic Web pages (HTML, XML). Servlets are a Java-equivalent of the CGI scripts and can be thought of as a server-side counterpart to the client-side Java applets.

Servlets are Java classes that implement a specific interface and produce HTML in response to GET/POST requests.

In an OOM, a servlet is represented as a component, it is linked to a servlet class that implements the required interface and provides the servlet implementation.

When you set the type of the component to Servlet, the appropriate servlet class is automatically created, or attached if it already exists. The servlet class is initialized so that operations are automatically added.

Servlet page of the component

When you set the type of the component to Servlet, the Servlet page is automatically displayed in the component property sheet.

The Servlet page in the component property sheet includes the following properties:

Property	Description	Maximum length
Servlet class	Class that implements the required interface. You can click the Properties tool beside this box to display the property sheet of the class, or click the Create tool to create a class	254
Servlet type	HttpServlet supports the http protocol, it is the most commonly used. GenericServlet extends the servlet generic class. User-defined implies some customization as it does not implement anything. The methods vary if you change the servlet type value	—

Defining servlet classes

Servlet classes are identified using the <<ServletClass>> stereotype.

The servlet class name is synchronized with the component name following the convention specified in the Value box of the Settings/Namings/ServletClassName entry in the Java object language.

Creating a servlet with the wizard

You can create a servlet with the **wizard** that will guide you through the creation of the component. The wizard is invoked from a **class diagram**. It is only available if the language is Java.

You can either create a servlet without selecting any class, or select a class beforehand and start the wizard from the contextual menu of the class.

You can also create several servlets of the same type by selecting several classes at the same time. The wizard will automatically create one servlet per class. The classes you have selected in the class diagram become servlet classes. They are renamed to fit the naming conventions standard, and they are linked to the new servlet component.

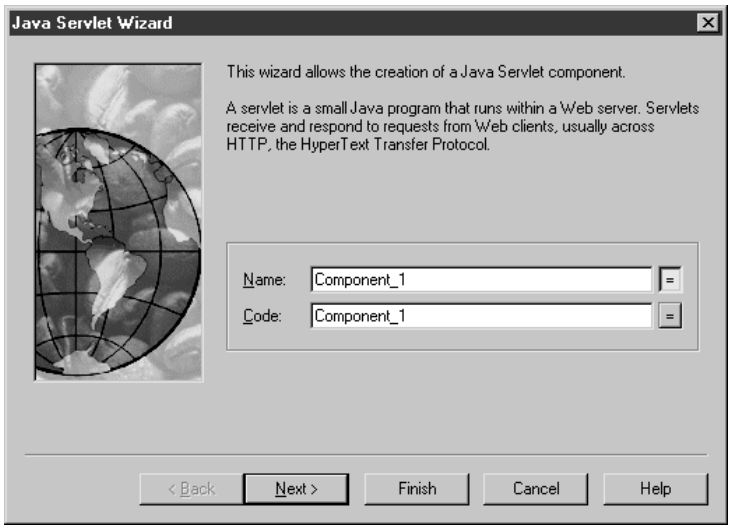
The wizard for creation of a servlet lets you define the following parameters:

Wizard page	Description
Name	Name of the servlet component
Code	Code of the servlet component
Servlet type	You can select the following types: HttpServlet that supports the http protocol (most commonly used), GenericServlet that extends the servlet generic class, or user-defined that implies some customization as it does not implement anything
Servlet class	Class that provides the Servlet implementation.
Create symbol	Creates a component symbol in the diagram specified beside the Create symbol In check box. If a component diagram already exists, you can select one from the dropdown listbox. You can also display the diagram properties by selecting the Properties tool
Create Class Diagram for component classifiers	Creates a class diagram with a symbol for each class. If you have selected classes before starting the wizard, they are used to create the component. This option allows you to display these classes in a diagram

❖ **To create a servlet with the wizard:**

1 Select Tools→Create Servlet from a class diagram.

The Java Servlet Wizard dialog box appears.


Java Servlet Wizard
This wizard allows the creation of a Java Servlet component.
A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.
Name:
Component_1
Code:
Component_1
< Back
Next >
Finish
Cancel
Help

2 Select a name and code for the servlet component and click Next.

If you have selected several classes, this page is not displayed because the names of the components are created by default according to the names of the classes.

3 Select a servlet type and click Next.

4 Select the servlet class name and click Next.

5 At the end of the wizard, you have to define the creation of symbols and diagrams.

When you have finished using the wizard, the following actions are executed:

- A servlet component is created
- The servlet class is created and visible in the Browser. It is named after the original class if you have selected a class before starting the wizard
- If you have not selected a class beforehand, it is prefixed after the original default component name to preserve coherence
- Any diagrams associated with the component are created or updated

Understanding servlet initialization and synchronization

When creating a servlet, the initialization process instantiates the servlet class together with its methods.

The role of the synchronization is to maintain the coherence of the whole model whenever a change is applied. It occurs progressively between classes already attached to a component. PowerDesigner successively performs several actions to complete synchronization as the model is modified.

The initialization and synchronization of the servlet class works in a similar way as with Message Driven bean classes:

- When the servlet class is attached to a servlet component, implementation methods for operations defined in the javax.servlet.Servlet interface are added by default. This interface is the base interface for all servlets, it may be included in the code depending on the servlet type selected. For HttpServlet and GenericServlet, the servlet class being directly derived from a class that implements it, it does not need to reimplement the interface. On the contrary, for user-defined servlets, this interface is implemented. You can see it from the Preview page of the servlet class.
- Implementation methods are removed when the class is detached if their body has not been altered
- The actual set of predefined methods vary depending on the servlet type

You can use the Check Model feature at any time to validate your model and complement the synchronization by selecting the Check Model feature from the Tools menu.

Generating servlets

The generation process retrieves all classes used by the servlet components to generate the servlets.

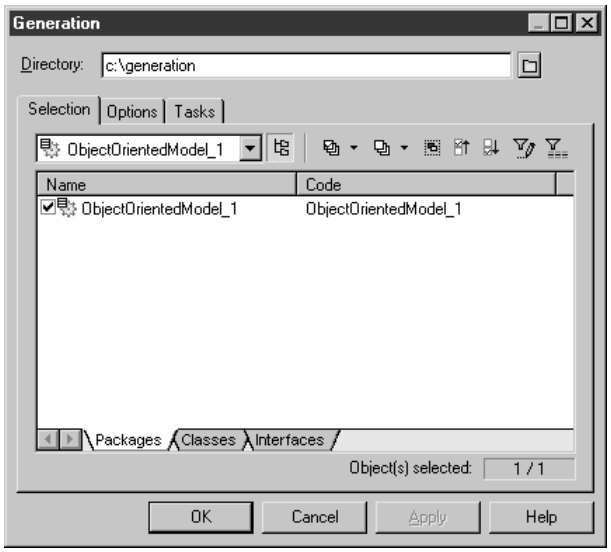
❖ **To generate servlets:**

1 Select Language→Generate Java Code to display the Generation dialog box.

2 Select or browse to a directory that will contain the generated files.

3 Click the Selection tab to display the Selection page.

4 Select the objects you need in the different tabbed pages.

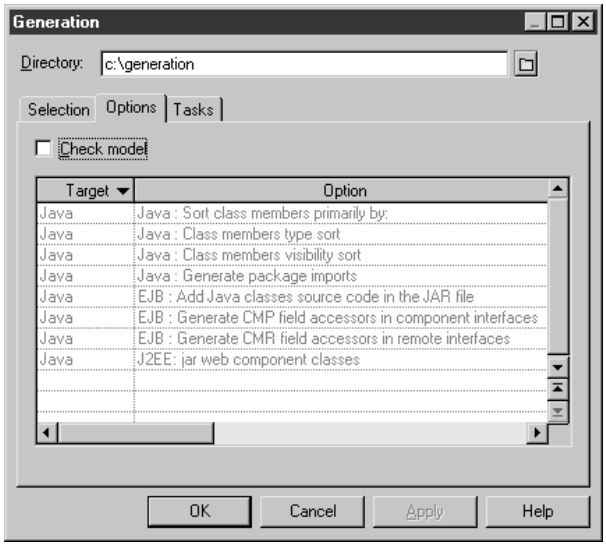


For more information on the selection page, see section Selecting objects to generate, in chapter Generating for a Language.

5 Click Apply.

6 Click the Options tab to display the Options page.

7 Specify your generation options in the Options page.

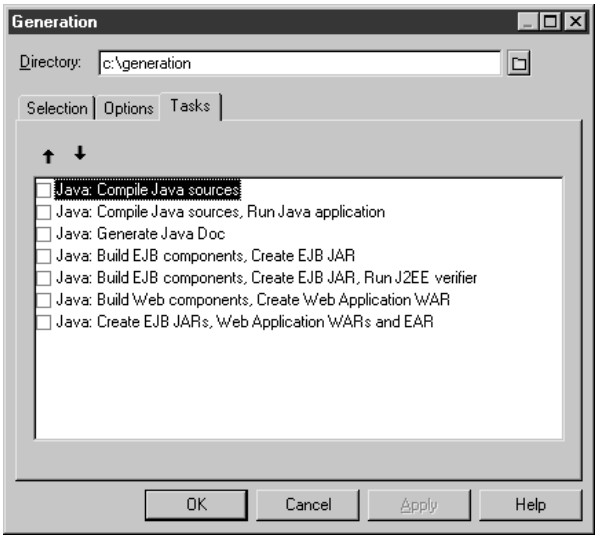

Generation
Directory: c:\generation
Selection Options Tasks
Check model
Target Option
Java Java : Sort class members primarily by:
Java Java : Class members type sort
Java Java : Class members visibility sort
Java Java : Generate package imports
Java EJB : Add Java classes source code in the JAR file
Java EJB : Generate CMP field accessors in component interfaces
Java EJB : Generate CMR field accessors in remote interfaces
Java J2EE: jar web component classes
OK Cancel Apply Help

For more information on the generation options, see section Defining generation options, in chapter Generating for a Language.

8 Click Apply.

9 Click the Tasks tab to display the Tasks page.

10 Select the commands you want to perform during generation in the Tasks page.



For more information on the generation tasks, see section Defining generation tasks, in chapter Generating for a Language.

You must beforehand set your environment variables from General Options→Variables in order to activate them in this page.

For more information on how to set these variables, see section Defining environment variables, in chapter Using the PowerDesigner interface, in the *General Features Guide*.

11 Click OK.

A progress box appears, followed by a Result list. You can use the Edit button in the Result list to edit the generated files individually.

12 Click Close.

The web.XML file is created in the WEB-INF directory and all files are generated in the generation directory.

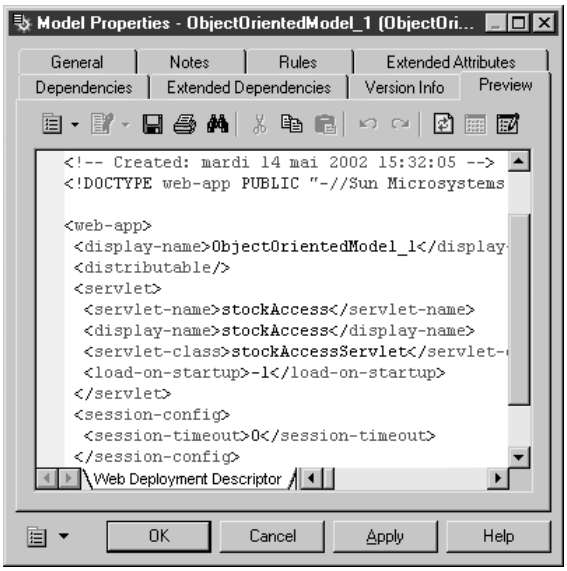
Generating servlet Web deployment descriptor

The Web deployment descriptor is an XML file, called web.XML. It is generated in the WEB-INF directory and is independent from the application server.

For more information on generating the Web deployment descriptor, see section Generating servlets.

The Web application deployment descriptor is generated per package. A WAR command available in the Tasks page of the Generation dialog box allows you to build a Web Archive that contains the Web deployment descriptor, in addition to all classes and interfaces referenced by servlet components. At the model level, an EAR command is also provided to group all WAR and JAR files generated for a given model inside a single enterprise archive. The EAR contains an additional deployment descriptor generated per model that is called application.XML.

The Web deployment descriptor contains several servlets to be deployed, it is available from the Preview page of the package, or the model property sheet.



Reverse engineering servlets

You can reverse engineer servlet code and deployment descriptor into an OOM. The reverse engineering feature reverses the Java class as a servlet class, it reverses the servlet as a component, and associates the servlet class with this component. The reverse engineering feature also reverses the deployment descriptor and all the files inside the WAR.

You start reverse engineering servlet from Language→Reverse Engineer Java. Select one of the following Java formats from the Selection page, and select the Reverse Engineer Deployment Descriptor check box from the Options page:

- Java directories
- Class directories
- Archive (.JAR file)

For more information on the Java formats, see section Reverse engineering Java formats in chapter Reverse engineering.

❖ **To reverse engineer servlets:**

1 Select Language→Reverse Engineer Java.

The Reverse Java dialog box appears.

2 Select Archive from the Reverse dropdown listbox in the Selection page.

3 Click the Add button.

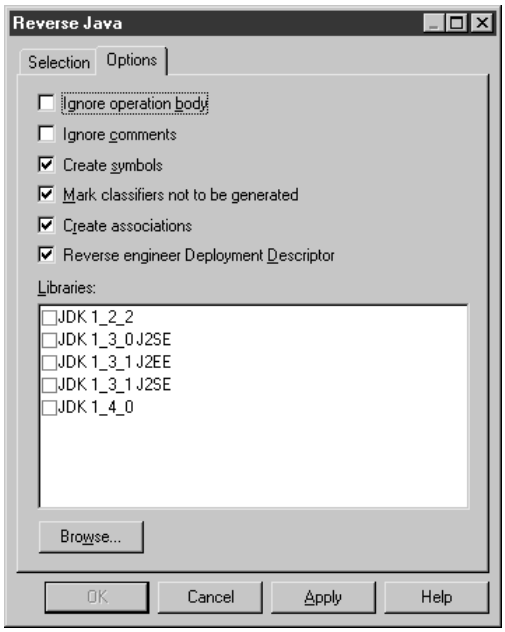
A standard Open dialog box appears.

4 Select the items you want to reverse and click Open.

The Reverse Java dialog box displays the items you selected.

5 Click the Options tab to display the options page.

6 Select the Reverse Engineer Deployment Descriptor check box.



7 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

8 Click OK.

The Reverse page of the Output window displays the changes that occurred during reverse and the diagram window displays the updated model.

Defining JSPs

Java Server Page (JSP) is an HTML Web page that contains additional bits of code that execute application logic to generate dynamic content.

In an OOM, a JSP is represented as a file object and is linked to a component - of type JSP -. The Java Server Page (JSP) component type allows you to identify this component. Components of this type are linked to a single file object that defines the page.

When you set the type of the component to JSP, the appropriate JSP file object is automatically created, or attached if it already exists. You can see the JSP file object from the Files page in the component property sheet.

JSP page of the component

When you set the type of the component to JSP, the JSP page is automatically displayed in the component property sheet.

The JSP page in the component property sheet includes the following properties:

Property	Description
JSP file	File object that defines the page. You can click the Properties tool beside this box to display the property sheet of the file object, or click the Create tool to create a file object
Default template	Extended attribute that allows you to select a template for generation. Its content can be user defined or delivered by default

To modify the default content, edit the current object language from Language→Edit Current Object Language and modify the following item: Profile/FileObject/Criteria/JSP/Templates/DefaultContent<%is(DefaultTemplate)%>. Then create the templates and rename them as DefaultContent<%is(<name>)%> where <name> stands for the corresponding DefaultContent template name.

To define additional DefaultContent templates for JSPs, you have to modify the JSPTemplate extended attribute type from Profile/Share/Extended Attribute Types and add new values corresponding to the new templates respective names.

For more information on the default template property, see the definition of TemplateContent in section Creating a JSP with the wizard.

Defining file objects for JSPs

The file object content for JSPs is based on a special template called DefaultContent defined with respect to the FileObject metaclass. It is located in the Profile/FileObject/Criteria/JSP/Templates category of the Java object language. This link to the template exists as a basis, therefore if you edit the file object, the link to the template is lost - the mechanism is similar to that of operation default bodies.

For more information on the Criteria category, see section Defining a criterion, in chapter Managing Profiles in the *Advanced User Documentation.*

Java Server Page files are identified using the JSPFile stereotype. The server page name is synchronized with the JSP component name following the convention specified in the Value box of the Settings/Namings/JSPFileName entry of the Java object language.

You can right-click a file object, and select Open With→ *text editor* from the contextual menu to display the content of the file object.

Creating a JSP with the wizard

You can create a JSP with the **wizard** that will guide you through the creation of the component. The wizard is invoked from a **class diagram**. It is only available if the language is Java.

You can either create a JSP without selecting any file object, or select a file object beforehand and start the wizard from the Tools menu.

You can also create several JSP of the same type by selecting several file objects at the same time. The wizard will automatically create one JSP per file object: the file objects you have selected in the class diagram become .JSP files.

The wizard for creation of a JSP lets you define the following parameters:

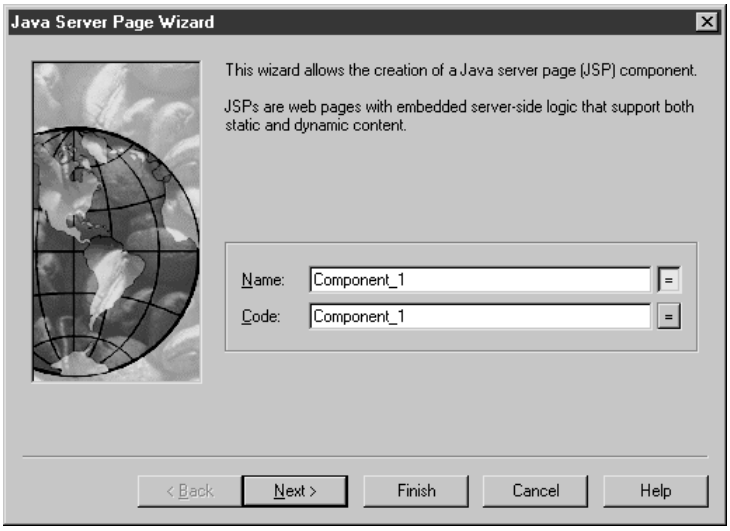
Wizard page	Description
Name	Name of the JSP component
Code	Code of the JSP component

Wizard page	Description
TemplateContent	Allows you to choose the default template of the JSP file object. The TemplateContent is an extended attribute located in the Profile/Component/Criteria/J2EE-JSP category of the Java object language. If you do not modify the content of the file object, the default content remains (see the Contents page of the file object property sheet). All templates are available in the Profile/FileObject/Criteria/JSP/templates category of the Java object language
Create symbol	Creates a component symbol in the diagram specified beside the Create symbol In check box. If a component diagram already exists, you can select one from the dropdown listbox. You can also display the diagram properties by selecting the Properties tool

❖ To create a JSP with the wizard:

1 Select Tools→Create JSP from a class diagram.

The Java Server Page Wizard dialog box appears.



2 Select a name and code for the JSP component and click Next.

If you have selected several file objects, this page is not displayed because the names of the components are created by default according to the names of the file objects.

3 Select a JSP template and click Next.

4 At the end of the wizard, you have to define the creation of symbols.

When you have finished using the wizard, the following actions are executed:

- A JSP component and a file object with an extension .JSP are created and visible in the Browser. The file object is named after the original default component name to preserve coherence
- If you open the property sheet of the file object, you can see that the Artifact property is selected

  For more information on artifact file objects, see section File object properties in a deployment diagram, in chapter Building a deployment diagram.
- You can edit the file object directly in the internal editor of PowerDesigner, if its extension corresponds to an extension defined in the Editors page of the General Options dialog box, and if the <internal> keyword is defined in the Editor Name and Editor Command columns for this extension

Generating JSPs

The generation process generates only file objects with the Artifact property selected.

Generating JSP Web deployment descriptor

The Web deployment descriptor is an XML file, called web.XML. It is generated in the WEB-INF directory and is independent from the application server.

The Web application deployment descriptor is generated per package. A WAR command available in the Tasks page of the Generation dialog box allows you to build a Web Archive that contains the Web deployment descriptor, in addition to all classes and file objects referenced by JSP components. At the model level, an EAR command is also provided to group all WAR and JAR files generated for a given model inside a single enterprise archive. The EAR contains an additional deployment descriptor generated per model that is called application.XML.

The Web deployment descriptor is available from the Preview page of the package, or the model property sheet.

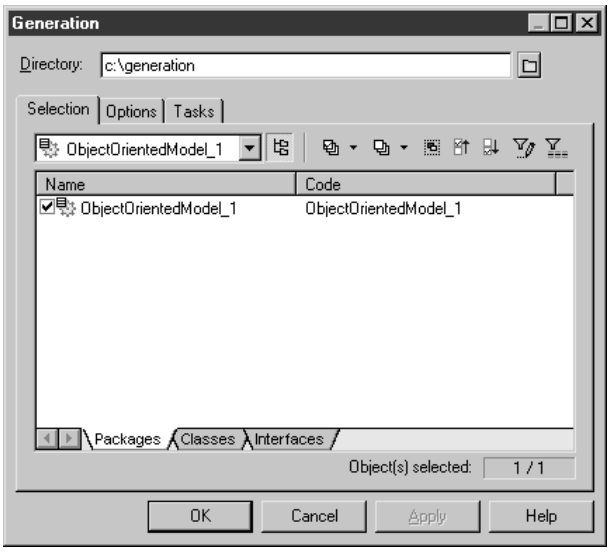
❖ **To generate JSPs:**

1 Select Language→Generate Java Code to display the Generation dialog box.

2 Select or browse to a directory that will contain the generated files.

3 Click the Selection tab to display the Selection page.

4 Select the objects you need in the different tabbed pages.

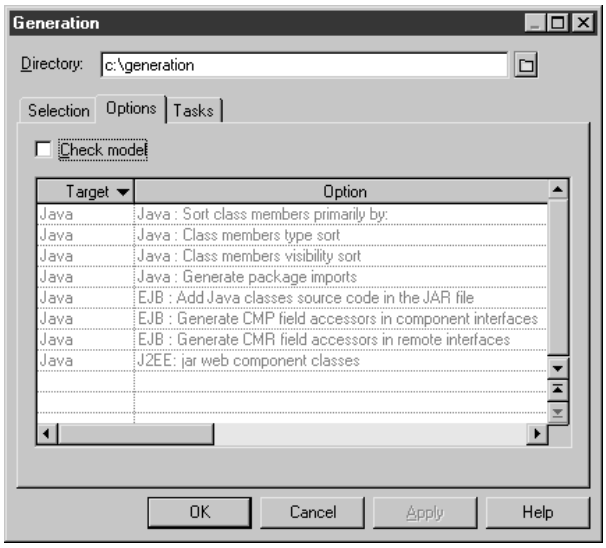


For more information on the selection page, see section Selecting objects to generate, in chapter Generating for a Language.

5 Click Apply.

6 Click the Options tab to display the Options page.

7 Specify your generation options in the Options page.

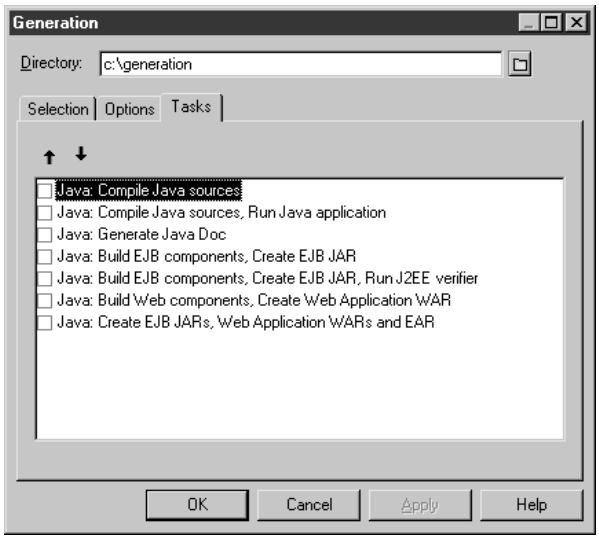

Generation
Directory: c:\generation
Selection Options Tasks
Check model
Target Option
Java Java : Sort class members primarily by:
Java Java : Class members type sort
Java Java : Class members visibility sort
Java Java : Generate package imports
Java EJB : Add Java classes source code in the JAR file
Java EJB : Generate CMP field accessors in component interfaces
Java EJB : Generate CMR field accessors in remote interfaces
Java J2EE: jar web component classes
OK Cancel Apply Help

For more information on the generation options, see section Defining generation options, in chapter Generating for a Language.

8 Click Apply.

9 Click the Tasks tab to display the Tasks page.

10 Select the commands you want to perform during generation in the Tasks page.


Generation
Directory: c:\generation
Selection Options Tasks
Java: Compile Java sources
Java: Compile Java sources, Run Java application
Java: Generate Java Doc
Java: Build EJB components, Create EJB JAR
Java: Build EJB components, Create EJB JAR, Run J2EE verifier
Java: Build Web components, Create Web Application WAR
Java: Create EJB JARs, Web Application WARs and EAR
OK Cancel Apply Help

For more information on the generation tasks, see section Defining generation tasks, in chapter Generating for a Language.

You must beforehand set the environment variables from General Options→Variables in order to activate them in this page.

For more information on how to set these variables, see section Defining environment variables, in chapter Using the PowerDesigner interface, in the *General Features Guide*.

11 Click OK.

A progress box appears, followed by a Result list. You can use the Edit button in the Result list to edit the generated files individually.

12 Click Close.

The web.XML file is created in the WEB-INF directory and all files are generated in the generation directory.

Reverse engineering JSPs

You can reverse engineer JSPs code and deployment descriptor into an OOM. The reverse engineering feature reverses the files to create JSP components and reverses the deployment descriptor inside the WAR.

You start reverse engineering JSPs from Language→Reverse Engineer Java. Select one of the following Java formats from the Selection page, and select the Reverse Engineer Deployment Descriptor check box from the Options page:

- Java directories
- Class directories
- Archive (.JAR file)

For more information on the Java formats, see section Reverse engineering Java formats in chapter Reverse engineering.

❖ To reverse engineer JSPs:

1 Select Language→Reverse Engineer Java.

The Reverse Java dialog box appears.

2 Select Archive from the Reverse dropdown listbox in the Selection page.

3 Click the Add button.

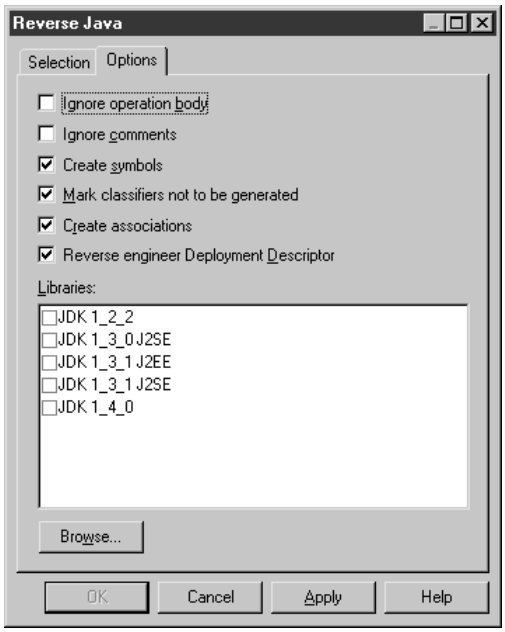
A standard Open dialog box appears.

4 Select the items you want to reverse and click Open.

The Reverse Java dialog box displays the items you selected.

5 Click the Options tab to display the options page.

6 Select the Reverse Engineer Deployment Descriptor check box.



7 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

8 Click OK.

The Reverse page of the Output window displays the changes that occurred during reverse and the diagram window displays the updated model.

Defining ASP.NET

An Active Server Page (ASP) is an HTML page that includes one or more scripts (small embedded programs) that are interpreted by a script interpreter (such as VBScript or JScript) and that are processed on a Microsoft Web server before the page is sent to the user. An ASP involves programs that run on a server, usually tailoring a page for the user. The script in the Web page at the server uses input received as the result of the user's request for the page to access data from a database and then builds or customizes the page on the fly before sending it to the requestor.

ASP.NET (also called ASP+) is the next generation of Microsoft's Active Server Page (ASP). Both ASP and ASP.NET allow a Web site builder to dynamically build Web pages on the fly by inserting queries to a relational database in the Web page. ASP.NET is different than its predecessor in two major ways:

- It supports code written in compiled languages such as Visual Basic, C#, and Perl
- It features server controls that can separate the code from the content, allowing WYSIWYG editing of pages

ASP.NET files have a .ASPX extension. In an OOM, an ASP.NET is represented as a file object and is linked to a component (of type ASP.NET). The component type Active Server Page (ASP.NET) allows you to identify this component. Components of this type are linked to a single file object that defines the page.

When you set the type of the component to ASP.NET, the appropriate ASP.NET file object is automatically created, or attached if it already exists. You can see the ASP.NET file object from the Files page in the component property sheet.

ASP page of the component

When you set the type of the component to ASP.NET, the ASP page is automatically displayed in the component property sheet.

The ASP page in the component property sheet includes the following properties:

Property	Description
ASP file	File object that defines the page. You can click the Properties tool beside this box to display the property sheet of the file object, or click the Create tool to create a file object
Default template	Extended attribute that allows you to select a template for generation. Its content can be user defined or delivered by default

To modify the default content, edit the current object language from Language→Edit Current Object Language and modify the following item: Profile/FileObject/Criteria/ASP/Templates/DefaultContent<%is(DefaultTemplate)%>. Then create the templates and rename them as DefaultContent<%is(<name>)%> where <name> stands for the corresponding DefaultContent template name.

To define additional DefaultContent templates for ASP.NET, you have to modify the ASPTemplate extended attribute type from Profile/Share/Extended Attribute Types and add new values corresponding to the new templates respective names.

For more information on the default template property, see the definition of TemplateContent in section Creating an ASP.NET with the wizard.

Defining file objects for ASP.NET

The file object content for ASP is based on a special template called DefaultContent defined with respect to the FileObject metaclass. It is located in the Profile/FileObject/Criteria/ASP/Templates category of the C# and VB.NET object languages. This link to the template exists as a basis, therefore if you edit the file object, the link to the template is lost - the mechanism is similar to that of operation default bodies.

For more information on the Criteria category, see section Defining a criterion, in chapter Managing Profiles in the *Advanced User Documentation.*

Active Server Page files are identified using the ASPFile stereotype. The server page name is synchronized with the ASP.NET component name following the convention specified in the Value box of the Settings/Namings/ASPFileName entry of the C# and VB.NET object languages.

You can right-click a file object, and select Open With→ *text editor* from the contextual menu to display the content of the file object.

Creating an ASP.NET with the wizard

You can create an ASP.NET with the **wizard** that will guide you through the creation of the component. The wizard is invoked from a **class diagram**. It is only available if the language is C# or VB.NET.

You can either create an ASP.NET without selecting any file object, or select a file object beforehand and start the wizard from the Tools menu.

You can also create several ASP.NET of the same type by selecting several file objects at the same time. The wizard will automatically create one ASP.NET per file object: the file objects you have selected in the class diagram become ASP.NET files.

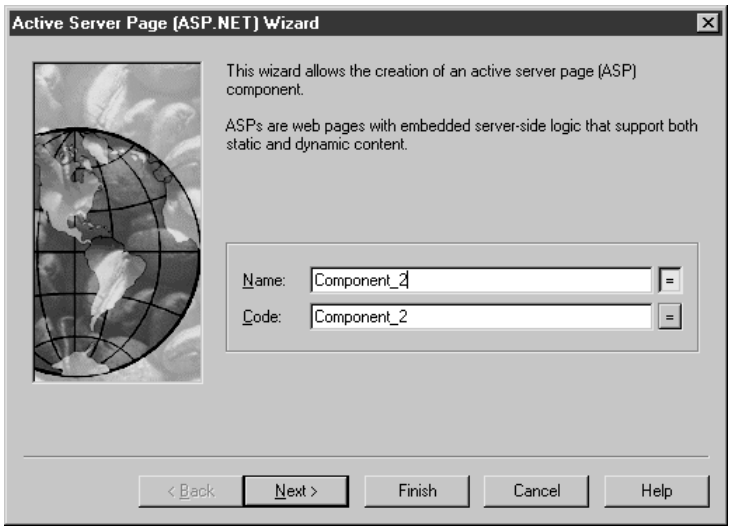
The wizard for creation of an ASP.NET lets you define the following parameters:

Wizard page	Description
Name	Name of the ASP.NET component
Code	Code of the ASP.NET component
TemplateContent	Allows you to choose the default template of the ASP.NET file object. The TemplateContent is an extended attribute located in the Profile/Component/Criteria/ASP category of the C# and VB.NET object languages. If you do not modify the content of the file object, the default content remains (see the Contents page of the file object property sheet). All templates are available in the FileObject/Criteria/ASP/templates category of the current object language
Create symbol	Creates a component symbol in the diagram specified beside the Create symbol In check box. If a component diagram already exists, you can select one from the dropdown listbox. You can also display the diagram properties by selecting the Properties tool

❖ **To create an ASP.NET with the wizard:**

1 Select Tools→Create ASP from a class diagram.

The Active Server Page Wizard dialog box appears.



2 Select a name and code for the ASP.NET component and click Next.

3 Select an ASP.NET template and click Next.

4 At the end of the wizard, you have to define the creation of symbols.

When you have finished using the wizard, the following actions are executed:

- An ASP.NET component and a file object with an extension .ASPX are created and visible in the Browser. The file object is named after the original default component name to preserve coherence
- If you open the property sheet of the file object, you can see that the Artifact property is selected

  For more information on artifact file objects, see section File object properties in a deployment diagram, in chapter Building a deployment diagram.

- You can edit the file object directly in the internal editor of PowerDesigner, if its extension corresponds to an extension defined in the Editors page of the General Options dialog box, and if the <internal> keyword is defined in the Editor Name and Editor Command columns for this extension

Generating ASP.NET

The generation process generates only file objects with the Artifact property selected.

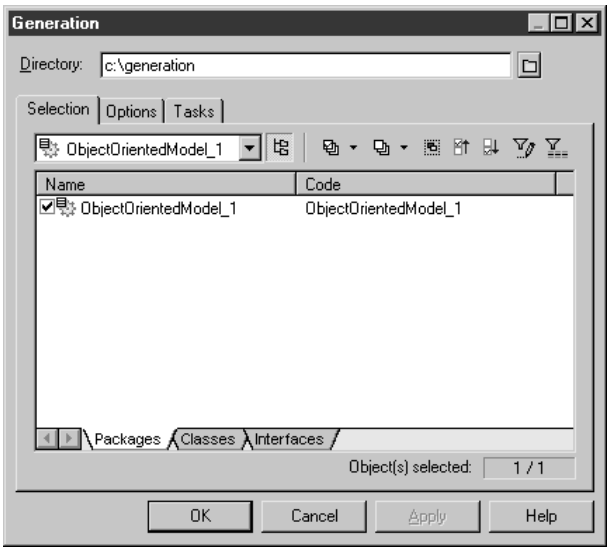
❖ **To generate ASP:**

1 Select Language→Generate C# or VB.NET code to display the Generation dialog box.

2 Select or browse to a directory that will contain the generated files.

3 Click the Selection tab to display the Selection page.

4 Select the objects you need in the different tabbed pages.

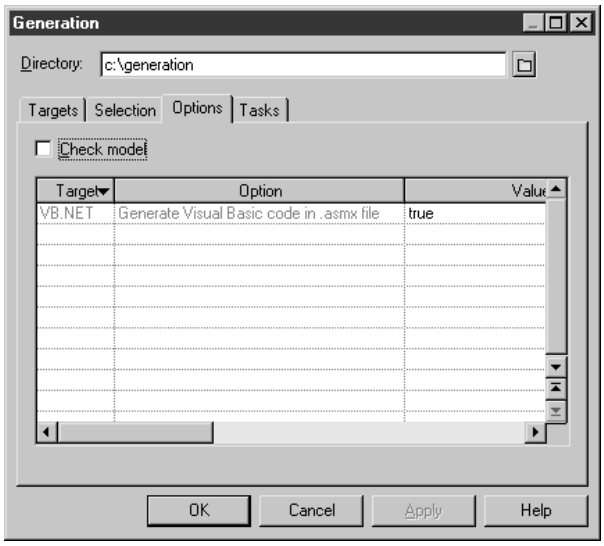


For more information on the selection page, see section Selecting objects to generate, in chapter Generating for a Language.

5 Click Apply.

6 Click the Options tab to display the Options page.

7 Specify your generation options in the Options page.

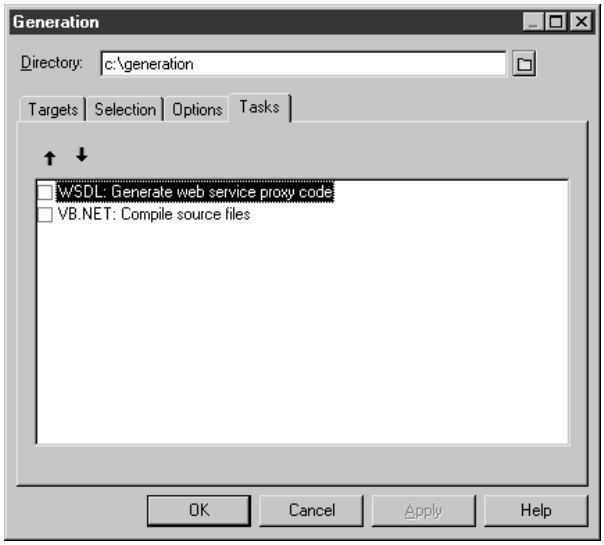


For more information on the generation options, see section Defining generation options, in chapter Generating for a Language.

8 Click Apply.

9 Click the Tasks tab to display the Tasks page.

10 Select the commands you want to perform during generation in the Tasks page.



For more information on the generation tasks, see section Defining generation tasks, in chapter Generating for a Language.

You must beforehand set the environment variables from General Options→Variables in order to activate them in this page.

For more information on how to set these variables, see section Defining environment variables, in chapter Using the PowerDesigner interface, in the *General Features Guide*.

11 Click OK.

A progress box appears, followed by a Result list. You can use the Edit button in the Result list to edit the generated files individually.

12 Click Close.

The files are generated in the generation directory.

CHAPTER 12

Working with Web Services

About this chapter

This chapter describes how to create and work with web services in an Object-Oriented Model (OOM).

Contents

Topic	Page
Defining Web services	470
Defining Web service classes	480
Defining Web service component instances	482
Generating Web services	485
Reverse engineering Web services	493

Defining Web services

A Web service is a service offered via the Web. The principle on which a Web service works is the following: a business application sends a request to a service at a given URL address using the SOAP protocol over HTTP. The service receives the request, processes it, and returns a response. An example that is most commonly used for a Web service is a stock quote service in which the request asks for the price of a specific stock and the response gives the stock price.

In an OOM, you design a Web service as a component (an EJB, a servlet, or a standard component can be a Web service) that includes a Web service implementation class.

When you work with web services in an OOM, you use the class, component and deployment diagrams. They allow you to do the following:

- Design and generate server side components for .NET and Java
- Generate client side proxy classes for.NET and Java
- Import components from a WSDL
- Browse a WSDL from UDDI
- Deploy Web services in a J2EE server
- Reverse engineer for .NET and Java

What do you need to work with Web services?

You need a Java, C# or Visual Basic .NET compiler.

You also need an XRPCC tool. XRPCC is utilized by the extended model definition named 'WSDL for Java'. XRPCC is a tool that generates WSDL and proxy for the Web service Java platform. It is delivered inside the WSDP (WebService Developer Pack) at: http://java.sun.com/webservices/webservicespack.html.

The WSDP must be installed on the client machine in addition to J2SE SDK 1.4 and J2EE SDK 1.3. These are available at the following Web sites: http://java.sun.com/j2ee/ and http://java.sun.com/j2se/

When you enter the WSDL environment variable from Tools→General Options, it must contain the path to the WSDL.EXE tool that is delivered by default with Microsoft Visual Studio.NET. It is a tool equivalent to XRPCC (used for proxy generation).

Audience

This chapter is primarily directed to anyone who is already familiar with Web services as it does not explain the required concepts and technologies in an exhaustive manner. It is intended to guide you through the use of PowerDesigner to help you create Web services and work with other Web services features.

Web services overview

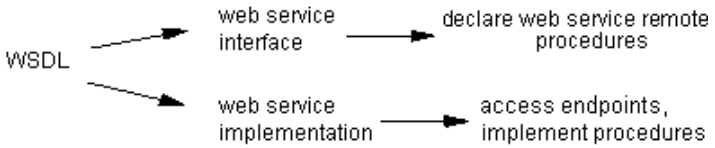
Web services operate over the Internet or a corporate intranet in the exact same way as you locate a Web site: either you type in the URL address or you use a search engine to locate the site.

With Web services, you may either know the address of the Web service you want to invoke or the address of its interface (WSDL for example), or you must search for the service by querying a Web service registry: it is the role of the UDDI specification that defines a standard mechanism for publishing and locating the existence of businesses and the services they provide.

What is WSDL?

WSDL stands for Web Service Description Language, it is an XML formatted document that describes what a Web service is capable of and how a client can locate and invoke that service.

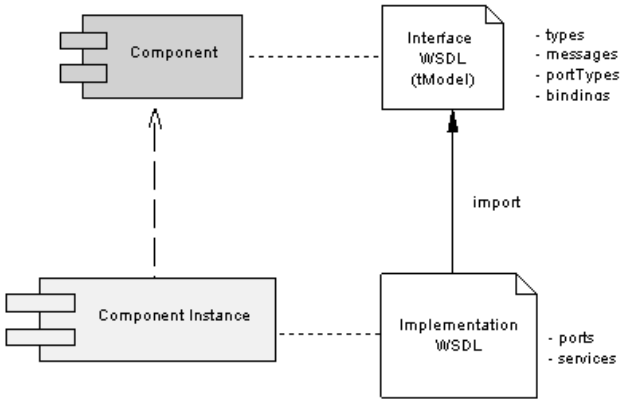
WSDL is used to define the Web service interface, the Web service implementation, or both. As a result, it is possible to use two WSDL files, one for the interface and one for the implementation.



In an **interface WSDL**, you declare the procedures that allow you to create a Web service.

In an **implementation WSDL**, you define how to implement these procedures through services and ports (access endpoints URLs).

In an OOM, an interface WSDL is associated with a component, and an implementation WSDL is associated with a component instance. You can save both WSDL files within the model.



What is SOAP?

SOAP stands for Simple Object Access Protocol. It is a protocol based upon XML for exchange of information in a distributed environment. It represents the invocation mechanism within a web service architecture.

What is UDDI?

UDDI stands for Universal Description Discovery and Integration. It is an XML-based registry for businesses worldwide. This registry lists all Web services on the Internet and handles their addresses.

In UDDI, an organization or a company, called a businessEntity usually publishes a WSDL to describe Web services interfaces as tModel. Another company may implement it, and then publish in the UDDI the following items that describe how to invoke the Web service:

- The company, called **businessEntity**
- The service, called **businessService**
- The access endpoints, called the **bindingTemplate**
- The specification, called the **tModel**

What is JAX-RPC?

JAX-RPC is a Java API for XML based RPC (Remote Procedure Calling protocol). It facilitates RPC over the Internet allowing XML formatted parameters to be passed to remote services and allowing XML formatted values to be returned. It is used in PowerDesigner.

What is JAXM?

JAXM is a Java API for XML messaging that provides a standard way to send XML documents over the Internet from the Java platform. It is used in PowerDesigner.

JSR specification

PowerDesigner follows the Web Services for J2EE specification (JSR109) that defines the programming model and runtime architecture for implementing web services.

Defining Web services targets

Supported languages

The following OOM languages support Web services:

- Java
- .NET: C# and Visual Basic .NET

In .NET, the WSDL can be generated by the service where the Web service is deployed. As a result, the WSDL generated by the .NET server contains both the interface and implementation definition.

In Java, only the implementation WSDL is generated by the server.

Not supported languages

The following OOM languages do not support Web services:

- Analysis
- PowerBuilder
- XML Schema
- XML DTD
- XML Data
- C++

Defining Web service component properties

A Web service is represented as a component that you can display in a component diagram. From a component diagram, you can display the Web service interface and implementation code, eventually, you can also deploy Web service components to nodes if you want to describe deployment of components into servers.

For more information on how to deploy components, see section Deploying a component to a node, in chapter Building a Component Diagram.

A component can be designed as a Web service Interface or a Web service Implementation.

You have to check the Web Service check box in the component property sheet to declare your component as a Web service. Once checked, the Web Service page and the WSDL page appear in the component property sheet.

Web Service page of the component

The Web Service page in the component property sheet includes the following properties:

Property	Description
Web service class	Web service class name. You can click the Properties tool beside this box to display the class property sheet, or click the Create tool to create a class. If the Web service component is a Stateless Session Bean or a Message Driven Bean, the Web service class is the bean class as well
Application namespace	Usually a package name within the server. The application namespace is used to generate the URL for the Web service in the server. By default, it uses the component package code or the model code, but you can modify it
WSDL URL	Indicates where the WSDL is published on the Web. For example: http://doc/Webservice1/StockQuote.asmx?WSDL or http://www.doc.com/Webservices/StockQuote.wsdl
Web service type	Interface or Implementation type. Interface refers to a component that defines the service interface only. Implementation refers to a component that implements a service interface
Use external WSDL	Indicates that the WSDL is published at a specific URL

Note

For Session Beans or Message Driven Beans, the Web service class and the bean class are synchronized; if you apply some changes in the EJB page, the Web Service page is automatically synchronized.

A Web service component may have an external WSDL if it is created from an external WSDL URL. In this case, the WSDL may be published by another organization, and you can either use it, or implement it. A Web service component created from a class can have its own WSDL.

You cannot change the WSDL URL of an external WSDL.

WSDL page of the component

The WSDL page in the component property sheet includes the following properties:

Property	Description
Target namespace	URL linked to a method that ensures the uniqueness of the Web service and avoid conflicts with other Web services of the same name. By default, it is `http://tempuri.org/` for .NET and `urn:%Code%Interface` for Java
Encoding style	You can choose the SOAP data types (soap:xxx) or the XML Schema data types (xsd:xxx) for the WSDL
Comment	Description of the WSDL file. It is used during WSDL generation
WSDL editor	You can use the text editor at the bottom of this page to edit the contents of the WSDL. When you click the User-Defined tool, you make the contents user defined. Once the tool is clicked, the contents can be overriden. When a WSDL is not user-defined, it is regenerated each time and can be displayed in the Preview page of the component property sheet

Preview page of the component

You can preview the code of a Web service component by clicking the Preview tab in the component property sheet. It allows you to preview the code of the following items:

- Web Service class and interfaces (for Session Bean, Servlet, and Message Driven Bean)
- .ASMX file (for .NET)

Display the component pages
You can display the Web Service, the WSDL, and the Code Preview pages of a component by right-clicking the component symbol in the diagram and selecting the appropriate page from the contextual menu.

Extended attributes page of the component

You can preview the extended attributes associated with a Web service component by clicking the Extended attributes tab in the component property sheet.

Creating a Web service with the wizard

You can create a Web service with the **wizard** that will guide you through the creation of the component. The wizard is invoked from a **class diagram** and is only available if you use the Java or the .NET family.

You can either create a Web service without selecting any class, or use the standard approach that consists in selecting an existing class first and then start the wizard from the contextual menu of the class.

You can also create several Web services of the same type by selecting several classes at the same time. The wizard will automatically create one Web service per class. The classes you have selected in the class diagram become Web service classes, they are renamed to fit the naming conventions standard, and they are linked to the new Web service component.

Using a namespace
You must create the Web service component within a package so that the package acts as a namespace.

The wizard for creation of Web services lets you define the following parameters:

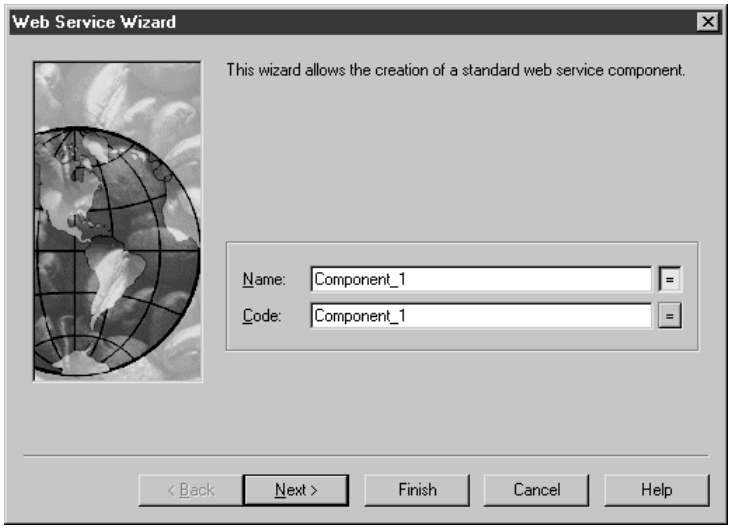
Property	Description
Name	Name of the Web service component
Code	Code of the Web service component
Web service type	Interface or Implementation. Interface refers to a component that defines the service interface only. Implementation refers to a component that implements a service interface
Component type	Standard, Stateless Session Bean, Servlet or Message Driven Bean (for languages other than Java, the only component type available is Standard) For more information on the different types of EJB, see section Using EJB types in chapter Working with platform dependent components
Web service implementation class	Defines the class that is used to implement the Web Service
Create symbol	Creates a component symbol in the diagram specified beside the Create symbol In check box. If a component diagram already exists, you can select one from the dropdown listbox. You can also display the diagram properties by selecting the Properties tool

Property	Description
Create Class Diagram for component classifiers	Available only for message driven beans, stateless session beans and servlets. It creates a class diagram with a symbol for each class and interface. If you have selected classes and interfaces before starting the wizard, they are used to create the component. This option allows you to display these classes and interfaces in a diagram

❖ **To create a Web service with the wizard:**

1 Select Tools→Create Web Service Component from a class diagram.

The Web Service Wizard dialog box appears.



2 Select a name and code for the Web service component and click Next.

If you have selected several classes, this page is not displayed because the names of the components are created by default according to the names of the classes.

When you create a Web service with the wizard, you have to select the type of the Web service and the type of the component that is uses as a source.

The following table maps available component types to Web service types in Java. For example, choose a standard component to generate code for Web service interface and Web service implementation using the JAX-RPC:

Component type	Java service interface	Java service implementation	Used with
Standard	✔	✔	JAX-RPC
EJB Stateless Session Bean	—	✔	JSR109 spec.
EJB Message Driven Bean	—	✔	JSR109 spec.
Java Servlet	—	✔	JAXM

✔ = allowed

— = not allowed

3 Select a Web service type and a component type and click Next.

4 Select a Web service implementation class and click Next.

5 At the end of the wizard, you have to define the creation of symbols.

When you have finished using the wizard, the following actions are executed:

- A Web service component flagged as 'Web service' is created
- A Web service implementation class is created and visible in the Browser. It is named after the original class if you have selected a class before starting the wizard. If you have not selected a class beforehand, it is prefixed after the original default component name to preserve coherence
- Depending on the component type, required interfaces associated with the component are added

Using data types with Web services

Web Services use XSD (XML Schema data types) with possible extensions.

WSDL data type mappings

In an OOM, the data type maps include three maps named WSDL2Local Data Types, Local2SOAP Data Types, and Local2XSD Data Types.

- The WSDL2Local map is used in WSDL reverse engineering
- The Local2SOAP map is used to provide a default value for the WSDL data type (for SOAP encoding)
- The Local2XSD maps object language data types to WSDL data types (for XSD encoding)

Modifying WSDL data type

If you need to use a specific data type, you can modify the WSDL data type of an operation. You can also modify the WSDL data type of a parameter if it is a simple data type.

As long as the WSDL data type is not manually changed, it is synchronized with the local data type. You can use the Preview page in the property sheet of the class to verify the code at any time.

❖ **To modify a WSDL data type:**

1 From the Web Method page in the operation property sheet, select or type a new data type in the WDSL data type box.
or
From the parameter property sheet, select or type a new data type from the WSDL data type box.

2 Click Apply

Using complex data types

Java — In Java, a Web service class can have a **serialization** class to convert an object into text or binary format, a **de-serialization** class to convert text, XML or binary format into an object.

.NET — In .NET, a method can have a SOAP Extension class to handle the serialization and de-serialization for the method and to handle security of other SOAP extensions features. You can select a class or create a new class to be used as a SOAP Extension class.

Declaring data types in the WSDL

Classes used as data types are declared as Complex Types inside the <types>/<schema> section.

Defining Web service classes

At the level of the class diagram, Web services require an implementation class. It is the class that is made available for the Web service. You need the implementation class to implement the Web service.

An implementation class can only be associated with one Web service component.

Detail page of the class

The Detail page in the class property sheet includes the following properties:

Property	Description
Web service component	Web service component name linked to the Web service class. You can click the Properties tool beside this box to display the component property sheet
Serialization class	Class used to convert an object into a text or binary format. You can click the Properties tool beside this box to display the class property sheet
De-serialization class	Class used to convert a text or binary format into an object. You can click the Properties tool beside this box to display the class property sheet

Preview page of the class

You can preview the code of a class by clicking the Preview tab in the class property sheet. It allows you to preview the code of the following items:

- The Web Service implementation class in Java
- The .ASMX file in .NET
- The interface WSDL that comes from the component in Java and .NET (it is in read-only mode)

Defining Web service operation properties

When the Web Service method check box is selected in the operation property sheet, it indicates that the operation is a method of a Web service.

When this check box is selected, a page called Web Method automatically appears in the operation property sheet.

Web Method page of the operation

The Web Method page in the operation property sheet includes the following properties:

Property	Description
SOAP extension class	Used for .NET. At creation of the class, new default functions are added. You can click the Properties tool beside this box to display the class property sheet or click the Create tool to create a class
WSDL data type	Data type for the return type. It shows basic WSDL types and XML Schema types

Defining Web service parameter properties

The parameter property sheet includes the following property, specific to Web services:

Property	Description
WSDL data type	Shows basic WSDL types and XML Schema types for parameters

Defining Web service component instances

The deployment diagram is used with Web services to model the Web services deployment. This diagram is useful if you want to deploy a Web service into one or several servers, and if you want to know which Web service is deployed where, as deployment diagrams can display network addresses, access endpoints and access types.

A component instance defines the ports, the access type, and the access endpoint that is the full URL to invoke the Web service.

When the Web Service check box is selected in the property sheet of the component instance, it means that the component instance is an instance of a Web service component. A component instance that inherits from a component inherits its type: the type of the component is displayed in the property sheet of the component instance.

When the Web Service check box is selected, a Web Service page and a WSDL page automatically appear in the property sheet of the component instance.

Web Service page of the component instance

The Web service page in the property sheet of the component instance includes the following properties:

Property	Description
Access point URL	Displays the full URL to invoke the web service. It is a calculated value that uses the network address located in the node. You can also type your own URL by using the User-Defined tool to the right of the box. Once clicked, the URL can be overriden
WSDL URL	Indicates where the WSDL should be published on the Web. You can type your own URL by using the User-Defined tool to the right of the box. Once clicked, the URL can be overriden
Use external WSDL	Indicates that the WSDL is published at a specific URL

When a WSDL is not user-defined, it is regenerated each time and can be displayed in the Preview page of the component instance property sheet. An external WSDL has a user-defined WSDL text.

Access point URL examples

Here are some examples of the syntax used in .NET and Java:

For .NET, the default syntax is:

```
accesstype://machine_networkaddress:port/application_namespace/webservice_code.asmx
```

For example: http://doc.sybase.com:8080/WebService1/StockQuote.asmx.

For Java, the default syntax is:

```
accesstype://machine_networkaddress:port/application_namespace/webservice_code
```

For example: http://doc.sybase.com/WebService1/StockQuote.

Computed attributes AccessType and PortNumber for code generator & VBScript are computed from the access point URL. For example: `http, https`.

WSDL URL examples

For .NET, the default syntax is:

```
accesstype://machine_networkaddress:port/application_namespace/Webservice_code.asmx?WSDL
```

For Java, the default syntax is:

```
accesstype://machine_networkaddress:port/application_namespace/wsdl_file
```

WSDL page of the component instance

The WSDL page in the property sheet of the component instance includes the following properties:

Property	Description
Target namespace	URL linked to a method that ensures the uniqueness of the Web service and avoid conflicts with other Web services of the same name. By default, it is: `http://tempuri.org/` for .NET, and `urn:%Component.targetNamespace%` for Java
Import interface WSDL	When selected, means that the implementation WSDL imports the existing interface WSDL
Comment	Used as the description of the WSDL file during WSDL generation and WSDL publishing in UDDI
WSDL editor	You can also use a text editor below the Comment area to display the contents of the WSDL. When you click the User-Defined tool among the available tools, you make the contents user defined. Once clicked, the contents can be overriden

Display the component instance pages
You can display the Web Service, and the WSDL pages of a component instance by right-clicking the component instance symbol in the diagram and selecting the appropriate page from the contextual menu.

Using node properties

The node property sheet includes the following property, specific to Web services:

Property	Description
Network address	Address or machine name. For example: doc.sybase.com, or 123.456.78.9

Since a machine can be used for several services and each service may have a different access type, port number and path, the machine Network address is only used as a default value. You can redefine the real URL of each component instance in the property sheet of the component instance at any time.

Generating Web services

You can generate client side or server side implementation classes, interfaces, deployment descriptor, JAR, WAR, or EAR from the Generate *object language* command in the Language menu.

Web services **server side** code generation consists in generating the following items:

- Generate Web Service implementation classes and interfaces (Stateless Session Bean, servlet, Message Driven Bean, .NET class, etc...)
- Generate additional files (.ASMX for .NET)
- Generate Web Services deployment descriptor for Java using the JSR109 specification. The deployment descriptor is an XML file that must be in every WAR archive, it is named WEB.XML by convention and contains information needed for deploying a Web service

  For more information on the deployment descriptor, see the glossary at the end of this manual.
- Generate interface WSDL and implementation WSDL files (WSDL files can be generated separately because they can be used for UDDI or client applications)

In general, once a Web service is deployed, the server is capable of generating an implementation WSDL.

Web services **client side** code generation consists in generating proxy classes.

Generating server side code for .NET

The following items are generated in .NET languages (C# and VB.NET):

- An implementation class (C# or VB.NET) with special super class and WebMethod property for the methods
- A .ASMX file

When generating the server side code, you can use default options and commands that help you to automatically start generating with pre-defined characteristics.

Note

It is not necessary to define the super class (also known as WebService) for the Web service classes; if the WebService super class is not defined, the code generator adds it to the code.

Defining Web services generation options in .NET

You can select Web services generation options available for .NET languages by starting the Generate *object language* command in the Language menu.

C# generation options

The following options are available from the Options page in the Generation dialog box for C#:

Option	Description
Generate C# Web Service code in .ASMX	Generates the C# code in the .ASMX file

VB.NET generation options

The following options are available from the Options page in the Generation dialog box for VB.NET:

Option	Description
Generate VB.NET Web service code in .ASMX	Generates the Visual Basic.NET code in the .ASMX file

Defining Web service generation tasks in .NET

You can select Web services generation tasks available for .NET languages by starting the Generate *object language* command in the Language menu.

The following tasks are available from the Tasks page in the Generation dialog box for C# and VB.NET:

Option	Description
Compile source files	Compiles the generated code
Generate Web service proxy code	Generates the proxy class

Generating and deploying web services in .NET

The deployment of a Web service consists in copying the generated implementation class and .ASMX file into the directory of the Web server virtual folder.

The .ASMX is an ASP.NET file, it contains the code of the C# or VB.NET Web service class.

❖ **To generate and deploy a Web service in .NET:**

1 Generate the implementation file and the .ASMX file into a folder `<drive>:\<machine_name>\wwwroot\<app_namespace>`

The name of the folder can be the name of the Web service namespace (for example: StockService).

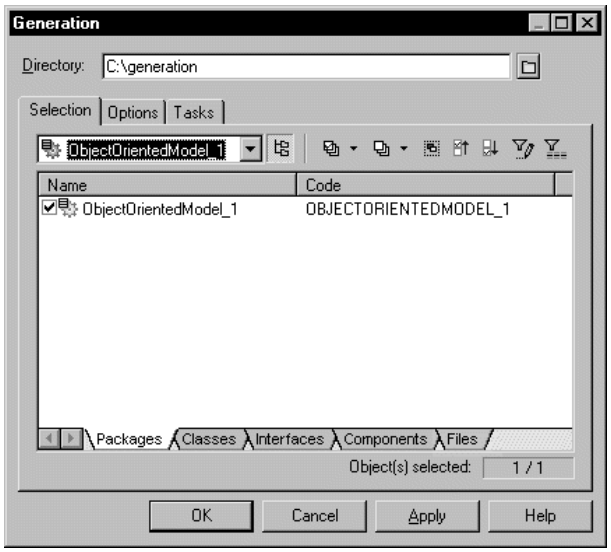
2 Invoke the Web service. The URL for the Web service is: `http:/<machine_name>/<app_namespace>/<service_name>.asmx`

The first time you invoke the .ASMX page, the server compiles the .ASMX page into .NET code.

3 Configure the Web service by adding or updating other files like Web.config, etc.

4 Select Language→Generate C# Code or Generate VB.NET Code to display the Generation dialog box.

5 Click the Selection tab to display the Selection page.



6 Select the `\<machine_name>\wwwroot` as the generation folder in the Directory box.

7 Select the parameters you need in the Options and Tasks pages (you do not need to compile).

8 Click OK.

The code generation process creates a subdirectory under wwwroot using the package name, and creates a <WebServiceName>.ASMX file within the subdirectory.

Generating server side code for Java

The following items are used for Web service implementation in Java:

- Servlets using JAXM to process SOAP message directly. JAXM is a Java API for XML messaging that provides a standard way to send XML documents over the Internet from the Java platform
- Standard Java class and interface that conform to the JAX-RPC definition. At deployment time, servlets are generated
- Stateless Session Bean supported by the Web Services for J2EE standard (JSR109)

JAX-RPC allows you to generate server and client side Artifact file objects for standard Java classes. You can use default command lines to generate, compile, package and deploy the Artifact file objects in the Generation dialog box available from the Generate Java code command in the Language menu.

The Web services deployment descriptor is generated using the JSR109 specification.

When generating the server side code, you can use default options and commands that help you automatically start generating with pre-defined characteristics.

Defining Web services generation options in Java

You can select Web services generation options available for Java by starting the Generate Java code command in the Language menu.

The following option is available from the Options page in the Generation dialog box for Java:

Option	Description
XRPCC input (WSDL)	Tool that generates WSDL and proxy for the Web service Java platform

Defining Web services generation tasks in Java

You can select Web services generation tasks available for Java by starting the Generate Java code command in the Language menu.

The following tasks are available from the Tasks page in the Generation dialog box for Java:

Option	Description
Generate Web service proxy code using XRPCC tool (client side)	Generates the proxy class using XRPCC
Generate WSDL using XRPCC tool (server side)	Generates the WSDL using XRPCC

Generating a WSDL

You can start the Generate WSDL command from the Language menu to generate the interface WSDL or the implementation WSDL. It is available from a class diagram, a component diagram or a deployment diagram. This opens the Generation dialog box from which you can select components and component instances.

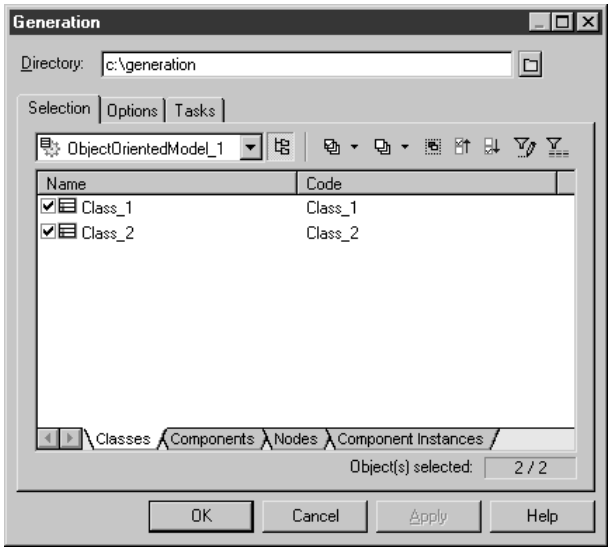
You can generate the interface or the implementation WSDL separately or with the code generation. Both WSDL files are generated from the component instance definition.

The WSDL template is different in .NET and in Java:

- In .NET, the template is very close to Microsoft format
- In Java, the template uses recommended format defined in JAX-RPC

❖ **To generate an interface or an implementation WSDL:**

1 Select Language→Generate WSDL to display the Generation dialog box.



2 Click the Selection tab to display the Selection page.

3 Select the objects to generate, using the required tabbed pages.

4 Click the Options tab to display the Options page.

The Options page may be empty by default, depending on your current source.

5 Click the Tasks tab to display the Tasks page.

6 Click OK to start generation.

A Progress box appears. The Result list displays the files that you can edit. The result is also displayed in the Generation page of the Output window, located in the bottom part of the main window.

7 Click Close to close the editor.

Generating Web services

In this section, you can find some details on generating Web services for the following types of components:

Generating servlet Web services

When a Web service component is a servlet, the http or SOAP processing is handled by the servlet, there is no additional code generated.

The generation, compilation, packaging and deployment processing is the same as with standard servlets.

Generating standard Web services

Generating standard Web service components consists in generating the following items:

- Generate the Java classes and interfaces
- Generate a configuration file for XRPCC
- Generate an Ant script file containing the tasks for generating server side code, compile the code, package and deploy the files. The deployment tasks depend on the target server
- Generate a WEB.XML file for the generated servlets
- Generate a .BAT file to launch the Ant script

In Java, a remote interface is generated for each Web service class. This interface exposes the Web methods of the class (conforming to the JAX-RPC definition).

The XRPCC compiler is used to generate the following items:

- The WSDL
- The server side Artifact file objects from the Java class
- The interface
- The configuration file

You can use the JAX-RPC compiler to generate server side code to avoid development and maintenance of the templates on your side.

Generating Stateless Session Bean Web services

Generating stateless Session Bean components is similar to generating EJBs. It consists in generating the following items:

- Generate the interface WSDL
- Generate the Web Services deployment descriptor for Java using the JSR109 specification

Creating Web service proxy components

You can create Web Service proxy components from the Tasks page in the Generation dialog box. This feature is available for Java, C# and VB.NET object languages.

❖ **To create a Web service proxy component:**

1 Select Language →Generate *Object Language* to display the Generation dialog box.

2 Click the Tasks tab to display the Tasks page.

3 Check the Generate web service proxy code check box to select this task.

4 Click OK to generate.

Generating client side code

You can launch the Generate *object language* command from the Language menu to generate the client side or server side implementation classes, interfaces, deployment descriptor, JAR, WAR, and EAR.

The proxy classes are generated if the model containing the definition of Web services components is opened.

In .NET, the client proxy code generation syntax is defined in the object language, it is based on the .NET Framework.

In Java, the client proxy code generation syntax is defined in the object language, it uses JAX-RPC.

Reverse engineering Web services

The process of reverse engineering Web services described in this section allows you to reverse WSDL for .NET and Java. This process is also defined as an import process, that is why it is introduced as the Import WSDL process.

Importing a WSDL

You can launch the Import WSDL feature from the Language menu. If you have a WSDL file or if you find a WSDL published in a UDDI server, you can reverse engineer the WSDL to create a Web service component.

When you launch the Import WSDL feature (from Language→Import WSDL), the Import WSDL dialog box appears.

Reverse engineering WSDL files

The Selection page in the Import WSDL dialog box contains the following items:

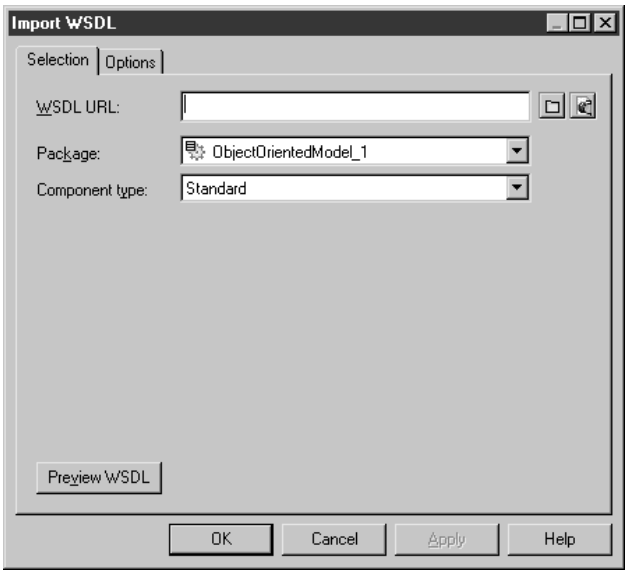
Item	Description
WSDL URL	Indicates where the WSDL should be published on the Web. You can either click the **Browse File** tool beside this box to select the file from a standard Windows file selection dialog box, or you can click the **Browse UDDI** tool beside this box to search for a WSDL file on a UDDI server. For more information on browsing UDDI, see section Browsing WSDL from UDDI
Package	Defines the package and namespace where the component and the Web service class are created
Preview WSDL	Allows you to preview the WSDL and the key (that is the unique key to locate the UDDI)
Component type	Displays the type of the component to create (for Java only)

Note If the WSDL contains an implementation, a component instance is created.

❖ **To reverse engineer WSDL files:**

1 Select Language→Import WSDL to display the Import WSDL dialog box.

2 Click the Selection tab to display the Selection page.



3 Select the WSDL file that you want to import.

4 Select a package.

5 <Java only> Select a component type.

6 Click Apply or OK to reverse.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

7 Click OK.

The reversed items are added to your model and are visible in the diagram. They are also listed in the Reverse page of the Output window, located in the lower part of the main window.

Reverse engineering WSDL options

The import process uses a WSDL file or a WSDL URL as an entry and allows you to create a component and its associated class. You can also create a deployment diagram with nodes and component instances to symbolize deployment.

The Options page in the Import WSDL dialog box includes the following items:

Item	Description
Create in component diagram	Allows you to create the symbol of components in the component diagram of your choice
Create in class diagram	Allows you to create the symbol of classes in the class diagram of your choice
Create in deployment diagram	Allows you to create the symbol of component instances in the deployment diagram of your choice

❖ **To define WSDL reverse engineering options:**

1 Select Language→Import WSDL to display the Import WSDL dialog box.

2 Click the Options tab to display the Options page.



3 Select the options you need.

4 Click Apply.

Browsing WSDL from UDDI

You can browse for a WSDL on a UDDI server. The Browse UDDI dialog box includes the following items:

Item	Description
UDDI operator URL	You can choose among a list of default UDDI operator URLs, or you can type URLs according to your needs
UDDI version	The request sent to the server is different depending on the UDDI version, you must choose the correct UDDI version for the URL
Search for	You can search for a name of a keyword that corresponds to the item selected in the Search In dropdown listbox
Search in	You can search for a Web service per business entity (company name), Web service name, WSDL name or tModel

The result is displayed in the Search Result window, it shows the Business entity name, the Service name, the Service description, and the WSDL URL.

Notes

After you browse UDDI, the data reversed into the OOM is stored into the Extended Attributes page of the component property sheet (in the WSDL tabbed page).

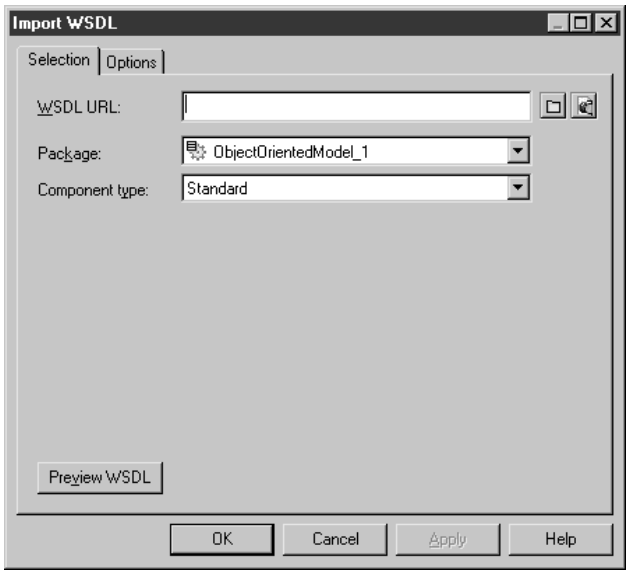
What do you need to browse UDDI?

You need to have Internet Explorer 5 or higher to use the Browse UDDI feature.

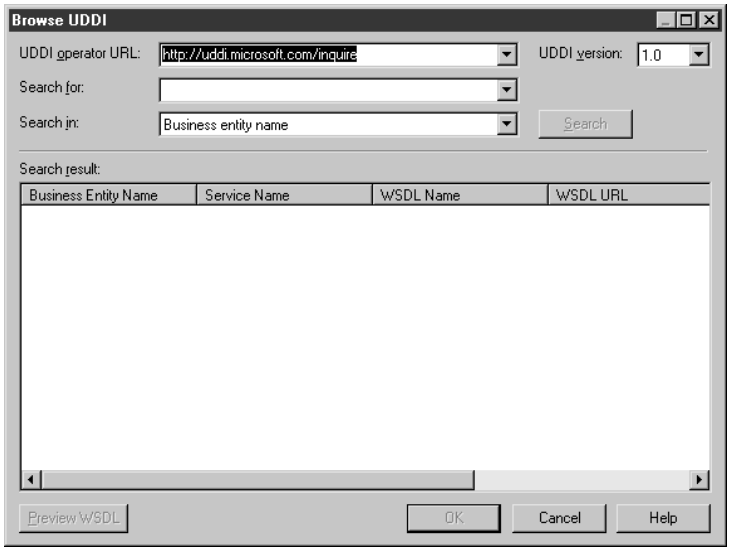
❖ **To browse WSDL files from UDDI:**

1 Select Language→Import WSDL to display the Import WSDL dialog box.

2 Click the Selection tab to display the Selection page.



3 Click the Browse UDDI tool beside the WSDL URL box to display the Browse UDDI dialog box.



4 Select a UDDI operator URL.

5 Select an item to search from the Search In dropdown listbox.

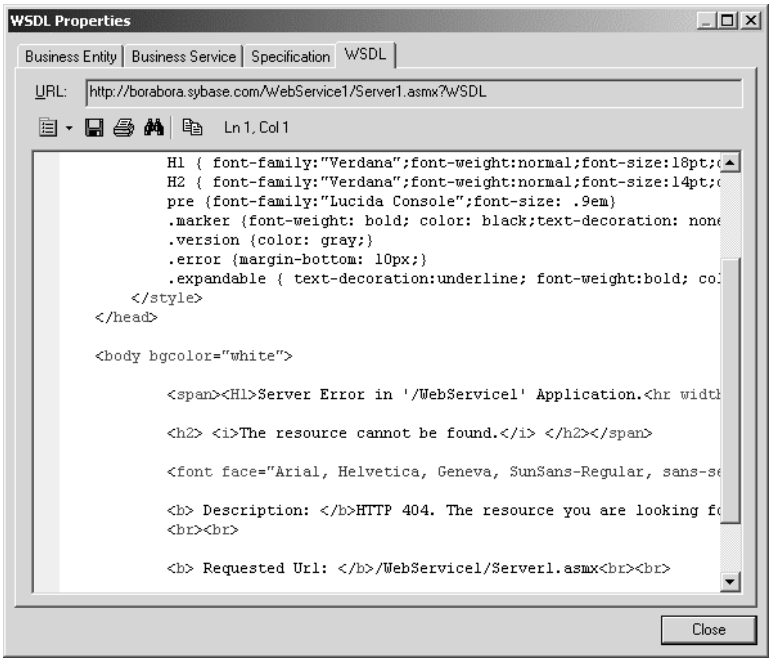
6 Type a keyword in the Search For dropdown listbox.

7 Click the Search button.

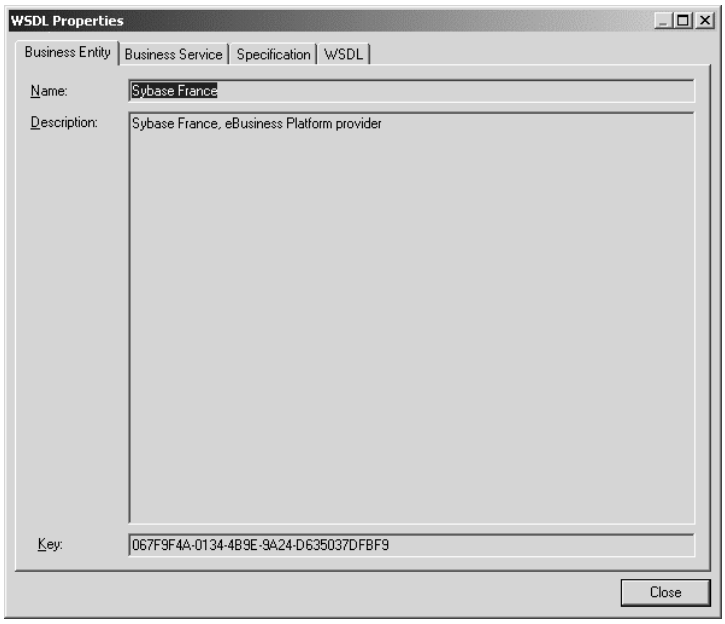
The result is displayed in the Search Result window.

8 Click the Preview WSDL button to open the WSDL property sheet.

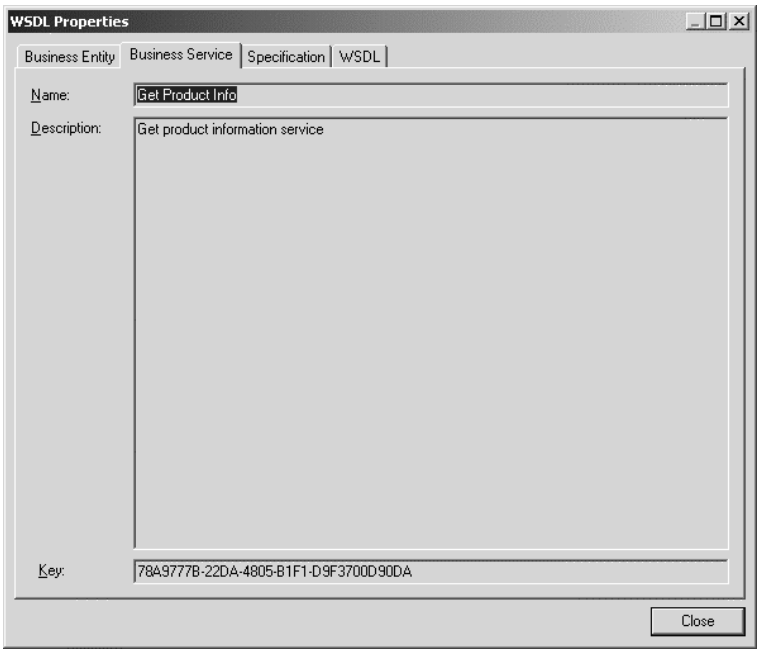
9 Click the WSDL tab to display the WSDL.



10 <optional> Click the Business Entity tab to display data about the company.



11 <optional> Click the Business Service tab to display data about the service.



12 Click Close to close the WSDL property sheet.

13 Click OK to go back to the Import WSDL dialog box.

For more information on the Import WSDL dialog box, see section Importing a WSDL

CHAPTER 13

Working with Object-Oriented Models

About this chapter

This chapter describes how to use diagrams in an Object-Oriented Model (OOM). It explains the interaction of multiple UML diagrams within one model and explains how they interact together.

It describes how to check the validity of an OOM: the Check Model feature allows you to verify the validity of OOM objects within a model at any time.

It also describes how to open a Rose model in an OOM.

Contents

Topic	Page
Defining related diagrams	502
Linking objects to other objects	505
Checking an OOM	516
Objects verified during Check model	523
Mapping objects in an OOM	560
Opening a Rose model in an OOM	574

Defining related diagrams

Each diagram of the OOM contains specific objects. However, when you are closer to implementation, if you want to attach existing objects to other types of diagrams (other than the one they were created in), you can use the **Related Diagrams** feature.

The Related Diagrams feature is available through a Related Diagrams page in the property sheet of OOM objects that can be linked to diagrams. This feature is used to detail the behavior, define the implementation of objects into other diagrams. It allows you to view objects from different angles and it participates in the definition of these objects. It describes a semantic relationship, and is not meant to explain in which diagrams an object can be found.

For examples on the use of related diagrams, see section Using related diagrams.

The Related Diagrams feature allows you to define, list, or remove relationships between a specific object and existing diagrams but it does not allow you to create new diagrams. You can associate any types of diagram of the OOM with those objects, including diagrams of other packages.

When you copy an object, the list of related diagrams appears empty. When you delete an object, you do not delete the related diagrams, you only detach them from the object.

Using related diagrams

Using related diagrams helps you explain in detail how some of the following OOM objects can be viewed from different angles.

For example, you may use related diagrams for the following reasons:

- Link a class diagram, collaboration diagram or sequence diagram to a use case to specify the implementation of a use case. When the use case is implemented this way in a class diagram for example and if you delete the use case, the class diagram is not deleted
- Specify a class decomposition, and show its links to use case diagrams. Even if you start your analysis with a use case diagram that is considered the highest conceptual level, you can go down to the class diagram, define the classes and go back to the use case diagram to further detail it

- Link a use case diagram to a use case to specify the use case decomposition, and show its relations to other use case diagrams. For example, a use case can be expanded into several other use case diagrams if the system needs to be further detailed
- Specify a class behavior in a collaboration or sequence diagram. For example, in a sequence diagram an object is an instance of a class represented in a time-based dynamic environment
- Describe the use of an operation in a collaboration diagram or a sequence diagram since both diagrams specify the behavior of classes and interfaces, and the possible use of their operations
- Specify a component behavior via a class diagram to help you detect the context in which a class is used to generate the component

Adding related diagrams

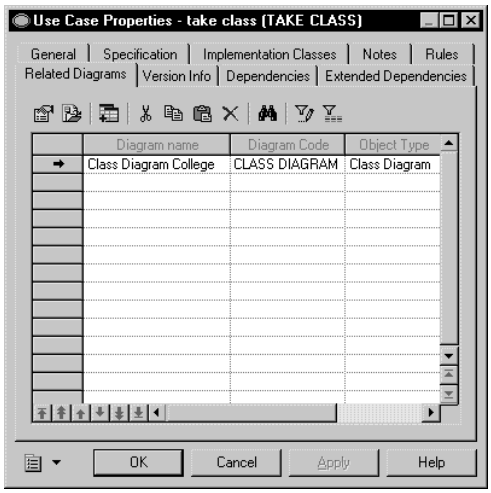
You can add related diagrams to OOM objects containing the Related Diagrams page in their property sheet.

❖ **To add related diagrams:**

1. Double-click the object symbol in the diagram to display the object property sheet.
2. Click the Related Diagrams tab to display the Related Diagrams page.
3. Click the Add Objects tool.

   A selection window appears.
4. Select the diagram you want to attach to the current object.
5. Click OK.

The selected diagram appears in the list of related diagrams for the current object.



Opening related diagrams

You can open any diagrams related to OOM objects containing the Related Diagrams page in their property sheet.

❖ **To open a related diagram from the object property sheet:**

1 Double-click the object symbol in the diagram to display the object property sheet.

2 Click the Related Diagrams tab to display the Related Diagrams page.

3 Select the diagram you want to open.

An arrow appears at the beginning of the line.

4 Click the Open Diagram tool.

The selected diagram opens in the diagram window, behind the object property sheet.

5 Click OK if you want to close the object property sheet.

Going back to the parent object
To go back to the parent object, double-click the parent diagram in the Browser to open the diagram and display the object symbol.

Linking objects to other objects

It is possible to link one object of a diagram type to another object of a different diagram type. You do that to show that one object evolves, through implementation or instantiation into another object. In this case, the evolution of the object is visible in two, or more diagrams of the model. It is also possible to attach objects between different Object-Oriented models.

You can attach an object to another object for the following reasons:

- Follow the projection of the object among different types of diagrams
- Show its implementation into one or more objects
- Show its instantiation into another object
- For documentation purposes also, you may want to follow up the behavior of an object to better identify its relationships with other objects, this may be helpful if you need to resume analysis of your system later.

You can attach objects in the following manner:

Object	Attached to	See section
Class and interface	Use case and actor	Linking a class or an interface to a use case and Linking a class or an interface to an actor
Use case, class and component	State	Linking a classifier to a state
Class	Object	Linking a class to an object
Component	Component instance	Linking a component to a component instance
Class	Object state	Linking a class to an object state
Object	Object state	Linking an object to an object state
Message	Operation (class)	Linking a message to an operation
Association	Instance link	Linking an association to an instance link

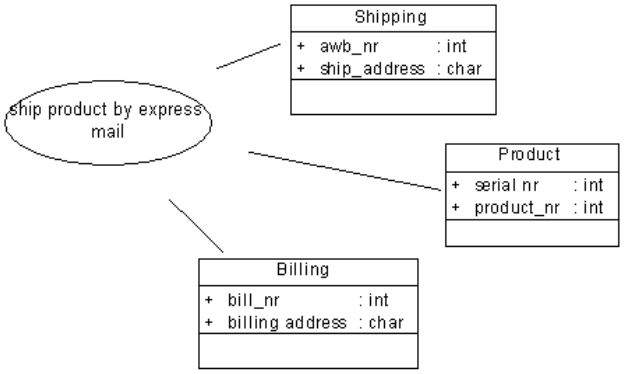
Linking a class or an interface to a use case

A use case can be used and expanded in class diagrams to describe how the use case is implemented by one or several classes, or one or several interfaces.

Implementation Classes

The Implementation Classes page in the use case property sheet shows the list of classes, or interfaces used to implement a use case. A use case is generally a task or service, represented as a verb. When analyzing what a use case must do, it is necessary to identify the classes and interfaces that need to be created to fulfill the task.
In this logic, it is possible to create the classes or interfaces and attach them to a use case.

For example, a use case **ship product by express mail** needs the classes **Shipping**, **Product**, and **Billing** to perform its task. They are shown as related to the use case in the figure below.



❖ **To link a class or an interface to a use case:**

1 Double-click the use case symbol to display its property sheet.

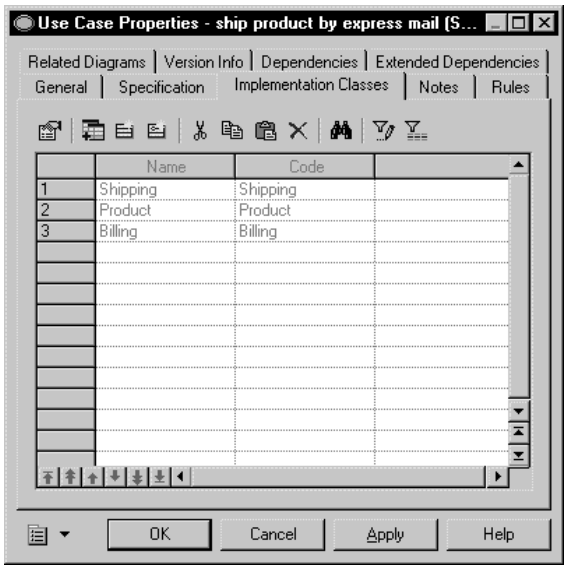
2 Click the Implementation Classes tab to view the list of classes.

3 Click the Add Objects tool.

A selection window appears listing the available classes and interfaces of the model.

Select Tools
All classes and interfaces of the model are displayed by default. Use the Include Sub-Packages tool beside the Folder Selection dropdown listbox to increase the selection, it allows you to include all the classes and interfaces from the packages of the model.

4 Select the objects as required and click OK.



The selected objects are displayed in the list of implementation classes.

You can similarly view the name of the use case in the class property sheet: double-click the class to open its property sheet, click the Dependencies tab and select the Use Cases tabbed page. The name of the use case is automatically added in this tabbed page.

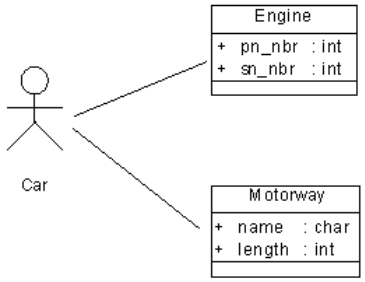
Linking a class or an interface to an actor

During the implementation phase, an actor could be implemented as one or several classes, or one of several interfaces.

Implementation Classes

The Implementation Classes page shows the list of classes and interfaces used to implement an actor. An actor can be a human being (person, partner) or a machine, or process (automated system). When analyzing what an actor must do, it is necessary to identify the classes and interfaces that need to be created for the actor to perform his task.
In this logic, it is possible to create the classes or interfaces and attach them to an actor.

For example, an actor Car needs the classes Engine and Motorway to perform its task. They are shown as related to the actor in the figure below.



❖ **To link a class or an interface to an actor:**

1 Double-click the actor symbol to display its property sheet.

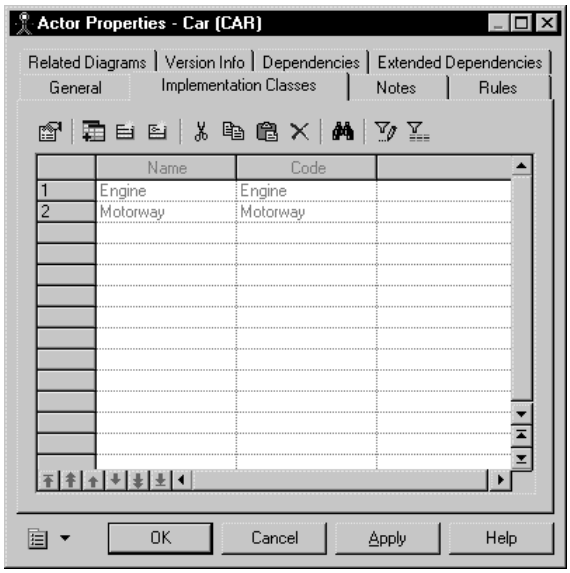
2 Click the Implementation Classes tab to view the list of implemented classes.

3 Click the Add Objects tool.

A selection window appears listing the available classes and interfaces of the model.

> **Select Tools**
> All classes and interfaces of the model are displayed by default. Use the Include Sub-Packages tool beside the Folder Selection dropdown listbox to increase the selection, it allows you to include all the classes and interfaces from the packages of the model.

4 Select the required objects and click OK.



The selected objects are displayed in the list of implementation classes.

You can similarly view the name of the actor in the class property sheet: double-click the class to open its property sheet, click the Dependencies tab and select the Actors tabbed page. The name of the actor is automatically added in this tabbed page.

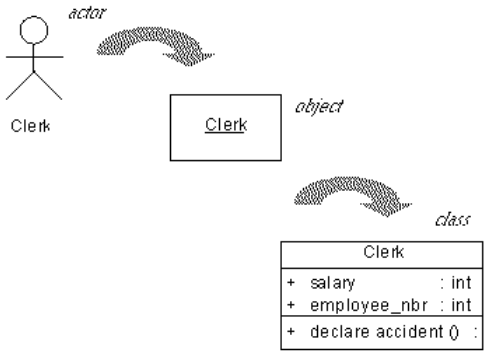
Example

Conceptually, you may link those elements even deeper. Here is an example of an actor in a use case diagram that becomes an object in a collaboration diagram. This object is then related to a class of a class diagram because it is an instance of this class:

A clerk works in an insurance company. He is an actor of a use case diagram, dealing with customers who declare a car accident.

The clerk actor becomes an object in a collaboration or sequence diagram; those diagrams representing the actions between the objects, the clerk as an object receives messages from customers and sends messages to his manager, for example.

The clerk as an object is also an instance of a class Clerk that can be found in a class diagram with its attributes and operations.



Linking a classifier to a state

When a statechart diagram is associated with a default classifier, the different states of the statechart diagram represent different situations during the life of this classifier. This classifier could be a class, a use case or a component that you can link to a state from the state property sheet.

❖ **To link a classifier to a state:**

1 Select a classifier from the Classifier dropdown listbox in the state property sheet.

The classifier name appears in the Classifier dropdown listbox.

2 Click OK.

The classifier is conceptually associated with the state.

You can similarly view the state name in the use case, class or component property sheet: click the Dependencies tab and select the States tabbed page. The state name is automatically added in this tabbed page.

Turning a classifier into a state
You can drag a use case, class or component from the Browser and drop it into the statechart diagram. You can also copy it and paste it, or paste it as shortcut, into the statechart diagram.

Linking a class to an object

The object diagram represents instances of class, the sequence diagram represents the dynamic behavior of a class, and the collaboration diagram represents those class instances in a collaboration mode. For all these reasons, you can link a class to an object in an OOM.

From the object property sheet, you can:

- Link the object to an existing class
- Create a new class

❖ **To link a class to an object:**

1 Select a class from the Class dropdown listbox in the object property sheet.
or
Click the Create tool beside the Class dropdown listbox to create a new class and display its property sheet.
Define the properties of the new class and click OK.

The class name appears in the Class dropdown listbox.

2 Click OK.

The object name is displayed in the sequence diagram, followed by a colon, and the name of the class selected.

You can similarly view the object name in the class property sheet: click the Dependencies tab and select the Objects tabbed page. The object name is automatically added in this tabbed page.

Turning a class into an object
You can drag a class node from the Browser and drop it into the sequence, collaboration or object diagrams. You can also copy a class and paste it, or paste it as shortcut, into these diagrams. This automatically creates an object, instance of the class.

Linking a component to a component instance

From a component instance property sheet, you can:

- Link a component instance to an existing component
- Create a new component

❖ **To link a component to a component instance:**

1 Select a component from the Component dropdown listbox in the property sheet of the component instance.
or
Click the Create tool beside the Component dropdown listbox to create a new component and display its property sheet.
Define the properties of the new component and click OK.

The component name appears in the Component dropdown listbox.

2 Click OK.

The component instance name is displayed in the deployment diagram.

You can similarly view the component instance name in the component property sheet: click the Dependencies tab and select the Component Instances tabbed page. The component instance name is automatically added in this tabbed page.

Turning a component into a component instance
You can drag a component node from the Browser and drop it into the deployment diagram. You can also copy a component and paste it, or paste it as shortcut, into the deployment diagram. This automatically creates a component instance, instance of the component.

Linking a class to an object state

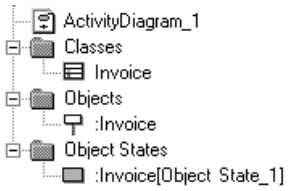
The activity diagram shows an object state as reflecting an object, instance of a class. For this reason, you can drag a class from the Browser and drop it into an activity diagram. This creates an object state (and an object) linked to the class.

❖ **To link a class to an object state:**

♦ Drag a class from the Browser and drop it into an open activity diagram.

This automatically creates an object and an object state in the Browser. The object state symbol is also displayed in the activity diagram, its name being the concatenation of the class and object state names.

Example: if you drag the class Invoice from the Browser and drop it into an activity diagram, you can see the following result in the Browser:



You can see the following object state symbol in the activity diagram:



For more information on object states, see section Defining object states, in chapter Building an activity diagram.

Linking an object to an object state

The activity diagram represents the object state as an object (instance of a class) in a particular state.

From the property sheet of the object state, you can:

- Link the object state to an existing object
- Create a new object

❖ **To link an object to an object state:**

1 Select an object from the Object dropdown listbox in the property sheet of the object state.
or
Click the Create tool beside the Object dropdown listbox to create a new object and display its property sheet.
Define the properties of the new object and click OK.

The object name appears in the Object dropdown listbox.

2 Click OK.

The name of the object state is displayed in the activity diagram, preceded by the name of the object.

You can similarly view the name of the object state in the object property sheet: click the Dependencies tab and select the Object states tabbed page. The name of the object state is automatically added in this tabbed page.

If you regularly use the Check Model feature, it is recommended to fill in the Object dropdown listbox in the object state property sheet with the name of an object, otherwise an error is displayed next time you run the Check Model feature.

Turning an object into an object state
You can drag an object from the Browser and drop it into an activity diagram. This automatically creates an object state. The object state symbol displays both object and object state names.

Linking a message to an operation

Linking a message to an operation during implementation can be very useful. If the receiver of a message is an object with an instantiated class, you can do the following from the message property sheet:

- Link the message to an existing operation
- Create a new operation

You cannot create an operation of an interface with this procedure. It only applies to operations of classes.

❖ **To link a message to an operation:**

1 Select an operation from the Operation dropdown listbox in the message property sheet.
or
Click the Create tool beside the Operation dropdown listbox to create a new operation and display its property sheet.
Define the properties of the new operation and click OK.

The operation name appears in the Operation dropdown listbox.

2 Click OK.

If you selected the Replace by Operation Name display preference, the message displays the name of the operation.

When you create a new operation, the name of the operation is also automatically added to the class and appears in the Operations page of the class property sheet.

Linking an association to an instance link

You can drag an association node from the Browser and drop it into the collaboration or object diagrams. This automatically creates two objects, and an instance link between them. Both objects are instances of the classes, and the instance link is an instance of the association.

Checking an OOM

You can check the validity of your OOM at any time using the Check Model feature from the Tools menu.

In the same way, the procedure of generating .java source files, PowerBuilder objects, or XML files starts by checking the validity of the OOM. If the model is not valid, or if an error is found, the files are not generated.

Use the Help command to select object control options
When you right click an OOM object control in the Check Model Parameters dialog box, a menu appears listing several options. The Help command opens a contextual help page explaining the checks performed for the selected object type.

OOM check options

When you check an OOM, if a parameter is found to be invalid, it will be displayed as an error or a warning in the Check Model window.

You can define levels of severity for problems that the Check Model feature finds, and you can have certain problems automatically corrected.

Levels of problem severity

You can identify the level of problem severity with the following icons.

Tool	Indicates	Description
	Error	Major problem that impedes Java or PowerBuilder generation
	Warning	Minor problem or recommendation

To display the severity level options, select Tools→Check Model, then expand the nodes of the tree view in the options tab.

These messages represent two different levels of problem severity. You can modify the level of problem severity for each object parameter verified by the Check model. This severity level can depend on the degree of normalization you want to achieve in your model.

Automatic correction

You can specify if you want to automatically correct an error using the Automatic Correction feature.

Tool	Indicates	Description
⊞	Automatic correction	PowerDesigner will correct the problem automatically

However, before using automatic correction, make sure you understand how it will affect your model.

Automatic correction is not available for all object parameters. Problems that cannot be corrected automatically must be corrected manually.

For more information on objects available for automatic correction, see section Objects verified during Check model.

Object selection in the Check Model

You select objects to be checked from the Selection page of the Check Model Parameters dialog box from Tools→Check Model.

You can list all objects, including composite objects created in the current model, current package, or sub-packages by selecting the Include Sub-Objects tool.

You have the following selection options:

Parent object	Include Sub-Objects selected	Include Sub-Objects not selected
Model	All objects created in model including all objects in packages and sub-objects	All objects created in model except objects in packages and sub-objects
Package	All objects created in package including all sub-objects	All objects created in package except sub-objects

Selecting objects in the diagram
If you graphically select objects in your diagram before starting the Check Model, they can automatically be selected for verification by the Check Model by clicking the Use Graphical Selection tool in the Selection page tool bar.

Checking an OOM

You can check the validity of an OOM at any time.

❖ **To check an OOM:**

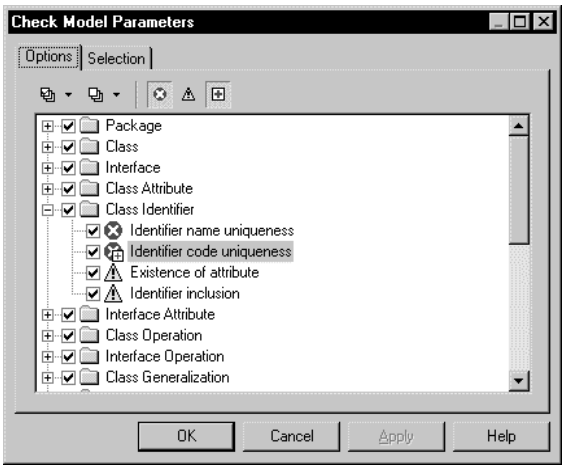
1 Select Tools→Check Model.
or
Right-click the diagram background and select Check Model from the contextual menu.

The Check Model Parameters dialog box appears.

2 Click the Options tab to display the Options page.

3 Expand an object parameter node.

The object parameters verified by the Check Model are displayed with the symbols indicating a degree of problem severity.



You want to	Action	Result
Modify a degree of problem severity	Select the object parameter, then the Error or Warning tool *Or* Right click the object parameter, select Error, Warning or Detail to get more information from the contextual menu	The icon changes to the appropriate severity level

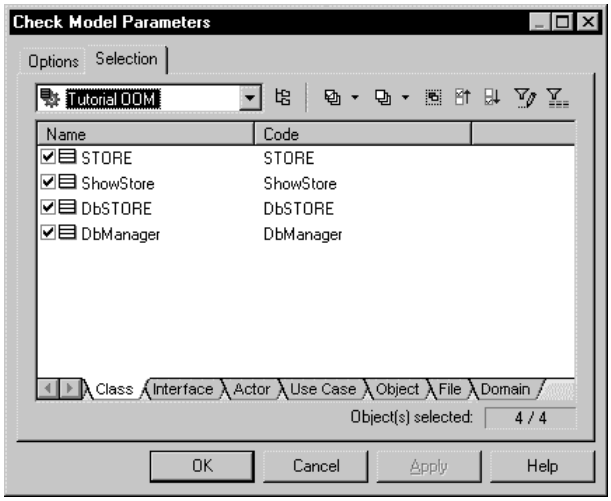
You want to	Action	Result
Set Automatic Correction of a problem	Select the object parameter, then the Automatic Correction tool *or* Right click the object parameter, select Automatic Correction from the contextual menu	The Automatic Correction icon is added to the Error or Warning symbol

4 Click the Selection tab to display the Selection page.

5 Select a model from the dropdown listbox at the top of the dialog box.

6 Click an object tab at the bottom of the Selection page.

The corresponding object page displays the objects of the model.

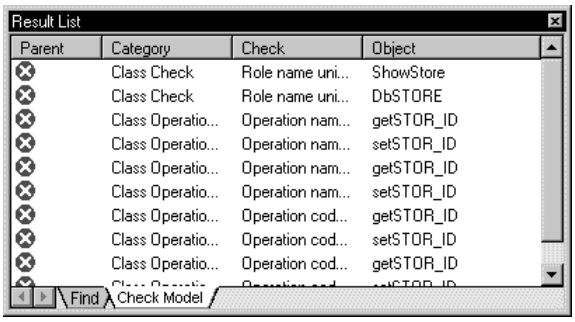


7 Select check boxes only for objects you want to be checked.

> **Selecting all or clearing all check boxes**
> You can select or clear all object check boxes by clicking the Select All tool, or the Deselect All tool.

8 Click OK.

The Check Model Result List displays errors and warnings based on the check options you have defined.



Dockable result window
When you right click an object parameter, a menu appears listing correction options. Among these, you can also select options to clear, dock or hide the result window.

Displaying previously applied check options

If you click the Apply button in the Check Model Parameters dialog box, all error and warning selections are stored in memory.

❖ **To display errors and warnings selected in the last check:**

♦ Select Tools→Check Model.

The Check Model Parameters dialog box opens to the Options page. The object parameters are displayed with the Error or Warning icons selected in the last check, or recorded.

Making corrections based on OOM check results

You use the Check Model feature to locate and correct problems in the OOM.

You use the following tool to set an automatic correction:

Tool	Indicates	Description
	Automatic correction	PowerDesigner will correct the problem automatically

You can choose the following correction options from the Check toolbar. If it is not displayed, select Tools→Customize Toolbars and select the Check check box.

Symbol	Option	Description
	Manual correction	Displays error and warning messages and property sheet of the problem object
	Check details	Error description and suggestion for correction
	Recheck Model	Checks selected object parameter, normally after a correction has been done
	Automatic correction	Automatically corrects: ♦ Non-unique names ♦ Code too long for generation (Java, PowerBuilder) ♦ Divergence in domain values, check parameters, and validation rules

Navigating in the error list

The Check toolbar contains navigation tools to move to the first, previous, next, or last error listed. You can also navigate in the list of errors by right-clicking an object parameter and selecting Go To First error, Previous error, Next error, or Last error from the contextual menu.

Right click menu
When you right click an object parameter a menu appears listing the correction options Manual Correction, Check Detail, Recheck, and Automatic Correction. You can also select options to clear, dock and hide the result window.

Making manual corrections to an OOM

Some errors cannot be corrected automatically, they have to be corrected manually.

❖ **To make manual corrections to an OOM:**

1 From the Result List, select an object parameter with error.

2 Right-click the object parameter and select Correct from the contextual menu to display the object property sheet.

3 Click the appropriate tab and make the necessary correction.

4 Close the property sheet.

5 Select the object parameter.

6 Right-click the object parameter and select Recheck from the contextual menu.

Verify that the problem has been corrected by starting Check Model again.

Making automatic corrections to the OOM

❖ **To make automatic corrections to an OOM:**

1 From the Result List, select an object parameter with error.

2 Right-click the object parameter and select Automatic Correction from the contextual menu.

3 Right-click the object parameter and select Re-check from the contextual menu.

Verify that the problem has been corrected by starting Check Model again.

Objects verified during Check model

The Check Model verifies the validity of the following objects in an OOM.

Domain check

During a domain check, the following controls are made.

Domain name and code uniqueness

Domain names and codes must only be used once in the model.

Manual correction	Automatic correction
Modify the duplicate domain name/code	Modifies the domain name or code of a selected object by appending a number to its current name

Inconsistency between default values and check parameters

Default values and validation rules (check parameters) must be consistent. Default values do not correct check parameters. Check parameters must also be defined consistently.

Manual correction	Automatic correction
Modify check parameters or default values to be consistent. Verify that check parameters are properly defined	—

File object check

During a file check, the following controls are made.

Embedded file name uniqueness

There cannot be two embedded files with the same name in a model.

Manual correction	Automatic correction
Rename one of the embedded files	Renames the file by appending a number to its name

Existence of external file location

External files must have a valid path location.

Manual correction	Automatic correction
Define a valid path location in the file property sheet	—

Replication check

During a replication check, the following controls are made.

Partial object replication

A replica object is partially synchronized with its replicated object.

Manual correction	Automatic correction
Modify the list of replicated attributes from the replication property sheet	Enforces the replication of desynchronized attributes of the replica object in the replication property sheet

Data source check

During a data source check, the following controls are made.

Data source name and code uniqueness

There cannot be two data sources with the same name or code in a model.

Manual correction	Automatic correction
Rename one of the data sources	Renames the data source object by appending a number to its name or code

Existence of physical data model

A data source must have at least one physical data model in its definition.

Manual correction	Automatic correction
Add a physical data model from the Models page of the property sheet of the data source	Deletes data source without physical data model

Differing DBMS types

The models in a data source represent a single database. This is why the models in the data source should share the same DBMS.

Manual correction	Automatic correction
Delete models with different DBMS or modify the DBMS of models in the data source	—

Package check

During a package check, the following controls are made.

Circular inheritance

Objects cannot be dependent on each other. Circular links must be removed.

Manual correction	Automatic correction
Remove circular generalization links	—

Circular dependency

Classes are dependent on each other through association class and/or generalization links. Circular links must be removed.

Manual correction	Automatic correction
Remove circular links	—

Shortcut code uniqueness

Two shortcuts with the same code cannot be in the same namespace.

Manual correction	Automatic correction
Change the code of one of the shortcuts	—

Actor check

During an actor check, the following controls are made.

Actor name and code uniqueness

Actor names and codes must be unique in the model or in the package.

Manual correction	Automatic correction
Change the name or code of the duplicate actor	Updates the actor name or code by appending a number to its current name or code

Isolated actor

An actor should not be isolated in the model.

Manual correction	Automatic correction
Create a link between the actor and a use case, or an object	—

Actor message check

During an actor message check, the following control is made.

Message between actors

An actor cannot send a message to another actor in the model. Messages are allowed between two objects, and between objects and actors.

Manual correction	Automatic correction
Create a message between two objects or between and actor and an object	—

Use case check

During a use case check, the following controls are made.

Use case name and code uniqueness

Use case names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate use case	Updates the use case name or code by appending a number to its current name or code

Isolated use case

A use case should not be isolated in the model.

Manual correction	Automatic correction
Create a link between the use case and an actor, or another use case	—

Class check

During a class check, the following controls are made.

Class name and code uniqueness

Class names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate class	Updates the class name or code by appending a number to its current name or code

Empty classifier

Attributes and operations are missing for this classifier.

Manual correction	Automatic correction
Add attributes or operations to the classifier	—

Persistent class with non-persistent attributes

All attributes of a persistent class cannot be non-persistent.

Manual correction	Automatic correction
Define at least one attribute as persistent	—

Association class with identifier

An associated class should not have identifiers.

Manual correction	Automatic correction
Remove the identifier	—

Class visibility

A private or protected class that is not inner to another class cannot be used.

Manual correction	Automatic correction
Remove private or protected visibility	Changes the visibility to public or package

Class constructor return type

A constructor cannot have a return type.

Manual correction	Automatic correction
Remove current return type of the constructor	Removes current return type of the constructor

Class constructor modifiers

A constructor cannot be static, abstract, or final.

Manual correction	Automatic correction
Remove the static, abstract, or final property of the constructor	Removes the static, abstract or final property of the constructor

Operation implementation

When there is a realization between a class and an interface, you must implement the operations of the interface within the class. To do so, click the Operations tab in the class property sheet and select the To be Implemented button at the bottom of the page to implement the missing operations.

Manual correction	Automatic correction
Implement the operations of the interface within the class	—

Role name assignment and uniqueness

A navigable role will be migrated as an attribute into a class. The code of the association is used if the role has no name.

Manual correction	Automatic correction
Assign a role name for the association role	—

The name of the role is used by another role or by another attribute.

Manual correction	Automatic correction
Change the name of the duplicate role	—

JavaBean without a BeanInfo

Bean implementors that provide explicit information about their beans must provide a BeanInfo class.

Manual correction	Automatic correction
Create a BeanInfo class	—

BeanInfo without a JavaBean class

A BeanInfo class must depend on a JavaBean class.

Manual correction	Automatic correction
Create a JavaBean class and recreate its BeanInfo, or delete the BeanInfo	—

Bean class definition

The Bean class must be defined as public. It must define a public constructor that takes no arguments and cannot define the finalize() method. It must be abstract for CMP Entity Beans but cannot be abstract or final for BMP Entity, Session and Message-driven Beans.

Manual correction	Automatic correction
Change the class visibility to public, define a public contructor with no arguments, do not define the finalize() method	Changes the class visibility to public, defines a public constructor with no arguments and removes the finalize() method. Corrects to set final = false, and set abstract = false

Bean class Business methods implementation

For each method defined in the component interface(s), there must be a matching method in the Bean class that has the same name, number, return type and types of arguments.

Manual correction	Automatic correction
Add a method with the same name, number, return type and types of arguments in the Bean class	Adds a method with the appropriate values in the Bean class

Bean class Home interface methods implementation

For each create<METHOD> method of the bean's Home Interface(s), there must be a matching ejbCreate<METHOD> method in the Bean class with the same method arguments. For each home method of the Home Interface(s), there must be a matching ebjHome<METHOD> method in the Bean class with the same number and types of arguments, and the same return type.

The following check applies to **Entity Beans** only.

For each ejbCreate<METHOD> method of the Bean class, there must be a matching ejbPostCreate<METHOD> method in the Bean class with the same number and types of arguments.

Manual correction	Automatic correction
Add a method with the same name and types of arguments in the Bean class	Adds a method with the appropriate values in the Bean class

The following check applies to **BMP Entity Beans** only.

For each find<METHOD> finder method defined in the bean's Home Interface(s), there must be a corresponding ejbFind<METHOD> method with the same number, return type, and types of arguments.

Manual correction	Automatic correction
Add a method with the same number, return type and types of arguments in the bean's Home Interface(s)	Adds a method with the appropriate values in the bean's Home Interface(s)

Bean class ejbCreate methods

ejbCreate<METHOD> methods must be defined as public, and cannot be final nor static.

The following check applies to **Entity Beans** only.

The return type of an ejbCreate() method must be the primary key type.

Manual correction	Automatic correction
Select the primary key in the Return type dropdown listbox of the Operation property sheet	Selects the primary key as return type

The following check applies to **Session Beans and Message Driven Beans.**

The return type of an ejbCreate() method must be void.

Manual correction	Automatic correction
Select void in the Return type dropdown listbox of the Operation property sheet	Changes the return type to void

The following check applies to **Message Driven Beans** only.

The Bean class must define an ejbCreate() method that takes no arguments.

Manual correction	Automatic correction
Add a method with no argument in the Bean class	Adds a method with no argument in the Bean class

Bean class ejbPostCreate methods

The following check applies to **Entity Beans** only.

ejbPostCreate<METHOD> methods must be defined as public, and cannot be final nor static. Their return type must be void.

Manual correction	Automatic correction
Change the method visibility to public, deselect the final and static check boxes and select void in the Return type dropdown listbox of the Operation property sheet	Changes the method visibility to public, changes the final and static check boxes and changes the return type to void

Bean class ejbFind methods

BMP Entity Bean specific.

ejbFind<METHOD> methods must be defined as public and cannot be final nor static. Their return type must be the entity bean's primary key type or a collection of primary keys.

Manual correction	Automatic correction
Change the method visibility to public and deselect the static check box	Changes the method visibility to public and deselects the static and final check boxes. Forces the return type of ejbFind<METHOD> to the primary key type

Bean class ejbHome methods

ejbHome<METHOD> methods must be defined as public and cannot be static.

Manual correction	Automatic correction
Change the method visibility to public and deselect the static check box	Changes the method visibility to public and deselects the static check box

Bean class ejbSelect methods

The following check applies to **CMP Entity Beans** only.

EjbSelect <METHOD> methods must be defined as public and abstract. Their throws clause must include the javax.ejb.FinderException.

Manual correction	Automatic correction
Change the method visibility to public, select the abstract check box, and include the javax.ejb.FinderException	Changes the method visibility to public, selects the abstract check box, and includes the javax.ejb.FinderException

Primary key class definition

The following check applies to **Entity Beans** only.

The primary key class must be declared as public and must define a public constructor that takes no arguments.

Manual correction	Automatic correction
Change the method visibility to public and add a default constructor in the primary key class	Changes the method visibility to public and adds a default constructor in the primary key class

Primary key class attributes

All primary key class attributes must be declared as public. In addition, each primary key class attribute must have a corresponding cmp-field in the Bean class.

Manual correction	Automatic correction
Change the visibility to public, and create a cmp-field in the Bean class that has the same name and the same data type as the attribute of the primary key class	Changes the visibility to public and creates a cmp-field in the Bean class that has the same name and the same data type as the attribute of the primary key class

Primary key class existence

If the bean class has more than one primary key attribute then a primary key class must exist. If there is only one primary key attribute, it cannot have a standard data type, but must have an object data type (ex: java.lang.Long).

Manual correction	Automatic correction
If there are many primary key attributes, create a primary key class. If there is only one primary key attribute, select an object/classifier data type	Creates a primary key class, or selects the appropriate object/classifier data type

Class mapping not defined

The class must be mapped to tables or views in the data source.

Manual correction	Automatic correction
Define the mapping from the Mapping page of the class property sheet. (Class Sources tab), or remove the data source	Removes the data source from the Mapping For dropdown listbox in the class Mapping page

For more information on O/R mapping, see section Understanding O/R mapping.

Identifier check

During an identifier check, the following controls are made.

Identifier name and code uniqueness

Identifier names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate identifier	Updates the identifier name or code by appending a number to its current name or code

Existence of attribute

Identifiers must have at least one attribute.

Manual correction	Automatic correction
Add an attribute to the identifier, or delete the identifier	—

Identifier inclusion

Two identifiers should not use the same attributes.

Manual correction	Automatic correction
Remove the unnecessary identifier	—

Interface check

During an interface check, the following controls are made.

Interface name and code uniqueness

Interface names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate interface	Updates the interface name or code by appending a number to its current name or code

Interface constructor

An interface cannot be instantiated. Thus a constructor cannot be defined for an interface.

Manual correction	Automatic correction
Remove the constructor	—

Interface navigability

Navigation is not allowed from an interface.

Manual correction	Automatic correction
Deselect navigability on the class side of the association	Deselects navigability on the class side of the association

Home interface create methods

The return type for create<METHOD> methods must be the bean's component interface type. The throws clause must include the javax.ejb.CreateException together with all exceptions defined in the throws clause of the matching ejbCreate<METHOD> and ejbPostCreate<METHOD> methods of the Bean class.

Manual correction	Automatic correction
Include the javax.ejb.CreateException and all exceptions defined in the throws clause of the matching ejbCreate<METHOD> and ejbPostCreate<METHOD> methods of the Bean class, or remove exceptions from the ejbPostCreate<METHOD> method	Includes the javax.ejb.CreateException and all exceptions defined in the throws clause of the matching ejbCreate<METHOD> and ejbPostCreate<METHOD> methods of the Bean class

Home interface finder methods

The return type for find<METHOD> methods must be the bean's component interface type (for a single-object finder) or a collection of primary keys thereof (for a multi-object finder). The throws clause must include the javax.ejb.FinderException.

Manual correction	Automatic correction
Include the javax.ejb.FinderException in the throws clause	Includes the javax.ejb.FinderException in the throws clause, and sets Return Type to be the component interface type

The following check applies to **BPM Entity Beans** only.

The throws clause must include all exceptions defined in the throws clause of the matching ejbFind<METHOD> methods of the Bean class.

Manual correction	Automatic correction
Include all exceptions defined in the throws clause of the matching ejbFind<METHOD> methods of the Bean class, or remove exceptions from the ejbFind<METHOD> method	Includes all exceptions defined in the throws clause of the matching ejbFind<METHOD> methods of the Bean class

Remote Home interface methods

The throws clause of the Remote Home interface methods must include the java.rmi.RemoteException.

Manual correction	Automatic correction
Include the java.rmi.RemoteException	Includes the java.rmi.RemoteException

Component interface business methods

The throws clause of the component interface business methods must include all exceptions defined in the throws clause of the matching method of the Bean class. The throws clause of the Remote interface methods must include the java.rmi.RemoteException.

Manual correction	Automatic correction
Include the java.rmi.RemoteException	Includes the java.rmi.RemoteException

Class Attribute check

During a class attribute check, the following controls are made.

Attribute name and code uniqueness

Attribute names and codes must be unique in the class.

Manual correction	Automatic correction
Change the name or code of the duplicate attribute	Updates the attribute name or code by appending a number to its current name or code

Inconsistencies within check parameters

The default value of the attribute (max, min, default or list of values) is not respecting the default validation rule (max, min, default or list of values), or is inconsistent.

Manual correction	Automatic correction
Correct check parameters in the Standard Checks page	—

Data type assignment

The data type of an attribute should be defined. Moreover, its type cannot be void.

Manual correction	Automatic correction
Assign a valid data type to the attribute	—

Initial value for final attribute

The final attribute of a class must be initialized.

Manual correction	Automatic correction
Give a default value to the final attribute	—

Domain divergence

The definition of the attribute definition is diverging from the definition of the domain.

Manual correction	Automatic correction
Modify attribute type to respect domain properties	Corrects attribute type to prevent divergence from domain

For more information on domain divergence, see section Defining OOM model options, in chapter Object-Oriented Model Basics.

Attribute mapping not defined

The attribute must be mapped to columns in the data source.

Manual correction	Automatic correction
Define the mapping from the Mapping page of the class property sheet. (Attributes Mapping tab)	Destroys the mapping for the attribute. This removes the attributes that are not mapped to any object in the Attributes Mapping page of the class Mapping page

For more information on O/R mapping, see section Understanding O/R mapping.

Interface Attribute check

During an interface attribute check, the following controls are made.

Attribute name and code uniqueness

Attribute names and codes must be unique in the interface.

Manual correction	Automatic correction
Change the name or code of the duplicate attribute	Updates the attribute name or code by appending a number to its current name or code

Inconsistencies within check parameters

The default value of the attribute (max, min, default or list of values) is not respecting the default validation rule (max, min, default or list of values), or is inconsistent.

Manual correction	Automatic correction
Correct check parameters in the Standard Checks page	—

Data type assignment

The data type of an attribute should be defined. Moreover, its type cannot be void.

Manual correction	Automatic correction
Assign a valid data type to the attribute	—

Initial value for final attribute

The final attribute of a class must be initialized.

Manual correction	Automatic correction
Give a default value to the final attribute	—

Domain divergence

The definition of the attribute definition is diverging from the definition of the domain.

Manual correction	Automatic correction
Modify attribute type to respect domain properties	Corrects attribute type to prevent divergence from domain

For more information on domain divergence, see section Defining OOM model options in Chapter Object-Oriented Model Basics.

Class Operation check

During a class operation check, the following controls are made.

Operation name and code uniqueness

Operation names and codes must be unique in the class.

Manual correction	Automatic correction
Change the name or code of the duplicate operation	Updates the operation name or code by appending a number to its current name or code

Return type and parameter data type assignment

The return type of an operation should be defined.

Manual correction	Automatic correction
Assign a return type to the operation	Assigns a void return type to the operation

The data type of a parameter should be defined. Moreover, its type cannot be void.

Manual correction	Automatic correction
Choose a valid data type for the parameter	—

Abstract operation body and non-abstract class

Abstract operations in a class cannot be implemented.

Manual correction	Automatic correction
Remove the body or the abstract property of the operation	—

Abstract operations must be declared in abstract classes only.

Manual correction	Automatic correction
Set the class to abstract, or remove the abstract property of the operation	Removes the abstract property in the operation property sheet

Overloading operations signature

Overloaded operations with the same name and same parameters data type cannot have different return types in a class.

Overloading an operation refers to using the same method name but performing different operations based on different parameter number or type.

Manual correction	Automatic correction
Change the operation name, parameter data type, or return type	—

Overriding operations

When overriding a parent operation in a class, it is impossible to change its modifiers.

Overriding an operation means that an operation defined in a given class is redefined in a child class, in this case the operation of the parent class is said to be overriden.

Manual correction	Automatic correction
Keep the same modifiers for child operation	—

Interface Operation check

During an interface operation check, the following controls are made.

Operation name and code uniqueness

Operation names and codes must be unique in the interface.

Manual correction	Automatic correction
Change the name or code of the duplicate operation	Updates the operation name or code by appending a number to its current name or code

Return type and parameter data type assignment

The return type of an operation should be defined.

Manual correction	Automatic correction
Assign a return type to the operation	Assigns a void return type to the operation

The data type of a parameter should be defined. Moreover, its type cannot be void.

Manual correction	Automatic correction
Choose a valid data type for the parameter	—

Realization check

During a realization check, the following controls are made.

Redundant realizations

Only one realization is allowed between two given objects.

Manual correction	Automatic correction
Remove redundant realizations	—

Class Generalization check

During a class generalization check, the following controls are made.

Redundant generalizations

Only one generalization is allowed between two given classes.

Manual correction	Automatic correction
Remove redundant generalizations	—

Multiple inheritance

The following check applies only to **Java** and **PowerBuilder**.

Multiple inheritance is accepted in UML but not in this language.

Manual correction	Automatic correction
Keep single inheritance	—

Extend final class

A final class cannot be extended.

Manual correction	Automatic correction
Remove the generalization link, or remove the final property in the parent class	—

Interface Generalization check

During an interface generalization check, the following controls are made.

Redundant generalizations

Only one generalization is allowed between two given interfaces.

Manual correction	Automatic correction
Remove redundant generalizations	—

Object check

During an object check, the following controls are made.

Object name and code uniqueness

Object names and codes must be unique in the model or in the package (in the same parent namespace) if they belong to the same class, or if they do not belong to any class.

Manual correction	Automatic correction
Change the name or code of the duplicate object	Updates the object name or code by appending a number to its current name or code

Isolated object

An object should not be isolated in the model.

Manual correction	Automatic correction
Create a relationship to or from the object. The relationship can be a message, an instance link, or a dependency *or* Link the object to an object state in the activity diagram	—

Note: the Check Model feature takes the object into account and not the symbol of the object to perform this check; if the object is already associated with an instance link or an object state in your model, the Check Model feature will not return an error message.

Instance link check

During an instance link check, the following controls are made.

Redundant instance links

Two instance links between the same objects should not have the same association.

Manual correction	Automatic correction
Remove one of the redundant instance links	—

Message check

During a message check, the following controls are made.

Message name and code uniqueness

Message names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate message	Updates the message name or code by appending a number to its current name or code

Message without sequence number

A message should have a sequence number.

Manual correction	Automatic correction
Enter a sequence number on the message	—

Message used by several instance links

A message should not be attached to several instance links.

Manual correction	Automatic correction
Detach the message from the instance link	—

State check

During a state check, the following controls are made.

State name and code uniqueness

State names and codes must be unique in the model or in the package for a given classifier. However, two states could share the same name and code if they are associated with distinct classifiers.

Manual correction	Automatic correction
Change the name or code of the duplicate state	Updates the state name or code by appending a number to its current name or code

Input transition missing

Each state must have at least one input transition. A state without an input transition cannot be reached.

Manual correction	Automatic correction
Add a transition linked to the state	—

Composite state does not have any start

A composite state details the state behavior in a sub-statechart diagram. To be complete, this sub-statechart diagram requires a start.

Manual correction	Automatic correction
Add a start in the sub-statechart diagram, or deselect the Composite check box in the state property sheet	—

Incorrect action order

The entry trigger events must be the first in the list of actions on a state. The exit trigger events must the last in the list. All other actions can be ordered without any constraint.

Manual correction	Automatic correction
Move all entry at the top of the list and all exit at the bottom	Moves all entry at the top of the list and all exit at the bottom

State action check

During an action check, the following controls are made.

Action name and code uniqueness

Action names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate action	Updates the action name or code by appending a number to its current name or code

Unspecified trigger event

Each action on a state must have a trigger event specified. This trigger event indicates when the action is executed.

Note that this check does not apply to actions defined on transitions because transitions have an implicit event corresponding to the end of execution of internal actions (of the source state).

Manual correction	Automatic correction
Specify a trigger event in the action property sheet	—

Duplicated occurrence

Two distinct actions of a same state should not occur simultaneously. The occurrence of an action is defined by combining a trigger event and a condition.

Manual correction	Automatic correction
Change the trigger event or the condition of the action	—

Event check

During an event check, the following controls are made.

Event name and code uniqueness

Event names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate event	Updates the event name or code by appending a number to its current name or code

Unused event

An event is useful to trigger an action defined on a state or on a transition. An event alone is useless.

Manual correction	Automatic correction
Delete the event or use it within an action on a state or on a transition	Deletes the event

Junction point check

During a junction point check, the following controls are made.

Junction point name and code uniqueness

Junction point names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate junction point	Updates the junction point name or code by appending a number to its current name or code

Incomplete junction point

A junction point represents a split or a merge of transition paths. That is why a junction point must have at least one input and one output transitions.

Manual correction	Automatic correction
Add any missing transitions on the junction point	—

Activity check

During an activity check, the following controls are made.

Activity name and code uniqueness

Activity names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate activity	Updates the activity name or code by appending a number to its current name or code

Input or Output transition missing

Each activity must have at least one input transition and at least one output transition.

Manual correction	Automatic correction
Add a transition linked to the activity	—

Composite activity without start

A composite activity details the activity execution in a sub-activity diagram. To be complete, this sub-activity diagram requires a start connected to other activities, or requires a start at the beginning.

Manual correction	Automatic correction
Add a start in the sub-activity diagram, or deselect the Composite check box in the activity property sheet	—

Decision check

During a decision check, the following controls are made.

Decision name and code uniqueness

Decision names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate decision	Updates the decision name or code by appending a number to its current name or code

Incomplete decision

A decision represents a conditional branch when a unique transition is split into several output transitions, or it represents a merge when several input transitions are merged into a unique output transition. That is why a decision must have more than one input transition or more than one output transition.

Manual correction	Automatic correction
Add any missing transitions on the decision	—

Object state check

During an object state check, the following controls are made.

Object state name and code uniqueness

Object state names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate object state	Updates the object state name or code by appending a number to its current name or code

Object state with undefined object

An object state represents a particular state of an object. That is why an object must be linked to an object state.

Manual correction	Automatic correction
In the property sheet of the object state, select or create an object from the Object dropdown listbox	—

Organization unit check

During an organization unit check, the following controls are made.

Organization unit name and code uniqueness

Organization unit names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate organization unit	Updates the organization unit name or code by appending a number to its current name or code

Circular dependency

The organization unit cannot be parent of itself or parent of another child organization unit.

Manual correction	Automatic correction
Change the organization unit in the Parent box in the organization unit property sheet	—

Start check

During a start check, the following controls are made.

Start name and code uniqueness

Start names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate start	Updates the start name, or code by appending a number to its current name or code

Missing transition

A start must be linked to an object of the statechart or activity diagram.

Manual correction	Automatic correction
Create a transition from the start	—

End check

During an end check, the following controls are made.

End name and code uniqueness

End names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate end	Updates the end name or code by appending a number to its current name or code

Missing transition

An end must be linked to an object of the statechart or activity diagram.

Manual correction	Automatic correction
Create a transition to the end	—

Synchronization check

During a synchronization check, the following controls are made.

Synchronization name and code uniqueness

Synchronization names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate synchronization	Updates the synchronization name or code by appending a number to its current name or code

Incomplete synchronization

A synchronization represents a fork when a unique transition is split into several output transitions executed in parallel, or it represents a join when several input transitions are joined and they wait until all transitions reach the join before continuing as a unique output transition. That is why a synchronization must have more than one input transition, or more than one output transition.

Manual correction	Automatic correction
Add any missing transitions to the synchronization	—

Transition check

During a transition check, the following controls are made.

Transition without source or destination

The transition has no source or destination. Select or create an object as source or destination.

Manual correction	Automatic correction
Assign a source or a destination to the transition	—

Useless condition

If there is only one output transition, there is no reason to have a condition on the transition.

Manual correction	Automatic correction
Remove the unnecessary condition, or create another transition with another condition	—

Missing condition

A transition must contain a condition if there are several output transitions on an activity or a decision, or when the transition is reflexive.

This check is slightly different if you work in a statechart diagram because an error occurs if no event or no condition has been defined on the transition.

Manual correction	Automatic correction
Define a condition on the transition, or create a synchronization to specify a parallel execution	—

Duplicated transition occurrence

You can use two parallel transitions (with same extremities) but they should not occur simultaneously. The occurrence of a transition is defined by conditions in the activity diagram and by both trigger event and conditions in the statechart diagram.

Manual correction	Automatic correction
Change the trigger event (statechart diagram only) or the condition of one transition	—

Component check

During a component check, the following controls are made.

Component name and code uniqueness

Component names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate component	Updates the component name or code by appending a number to its current name or code

Isolated component

A component should not be isolated in the model. It should be linked to a class or an interface.

Manual correction	Automatic correction
Attach some classes or interfaces to the component	—

EJB component attached classifiers

Entity and Session Beans must provide either a remote or a local client view, or both.

Manual correction	Automatic correction
Complete existing view(s), or create a remote view if no interface has been exposed	Completes existing view(s), or creates a remote view if no interface has been exposed

Node check

During a node check, the following controls are made.

Node name and code uniqueness

Node names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate node	Updates the node name or code by appending a number to its current name or code

Empty node

A node is said to be empty when it does not contain any component instance.

Manual correction	Automatic correction
Add at least one component instance to the node	—

Component instance check

During a component instance check, the following controls are made.

Component instance name and code uniqueness

Component instance names and codes must be unique in the model or in the package (in the same parent namespace).

Manual correction	Automatic correction
Change the name or code of the duplicate component instance	Updates the component instance name or code by appending a number to its current name or code

Component instance without component

An instance of a component has been created but no component is attached. You must attach it to a component.

Manual correction	Automatic correction
Attach an existing component to the component instance, or create a component from the Create tool in the property sheet of the component instance	—

Duplicate component instances

Several instances of the same component exist in the same node.

Manual correction	Automatic correction
Delete the duplicate component instance or attach it to the right component	—

Isolated component instance

A component instance should not be created outside of a node. It will not be deployed.

Manual correction	Automatic correction
Attach it to a node	—

Mapping objects in an OOM

Developers tend to use object-oriented programming languages like Java, to develop business objects and components. These objects can be stored in a database. A problem arises when the user tries to store objects in a relational database because object modeling describes a system through objects that have identity, behavior and encapsulated state whereas relational modeling describes a system by its data.

Furthermore, object codes in an OOM for a given object language are often different than codes used in a relational database. This requires that you modify object codes after generating an OOM into a PDM or a PDM into an OOM to be compliant with the object language.

You can use object mapping to bypass this impedance-mismatch.

Object mapping is the ability to establish a correspondence between objects belonging to heterogeneous models and diagrams. In an OOM, you can link classes to tables in a PDM, this process is called **object to relational mapping** (O/R mapping).

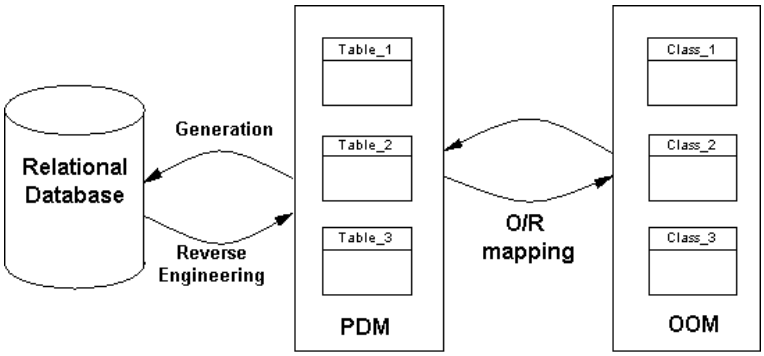
O/R mapping allows to manage the persistence of objects in a relational database.

Understanding O/R mapping

You create a mapping between OOM objects and PDM tables to setup object persistence. Data comes from a **data source** and is managed by an OOM. The mapping appears as a query defined in the OOM object: this query allows you to handle data in the data source.

In O/R mapping, the data source contains one or several physical data models representing the structure of the relational database in which OOM objects will be stored.

The following schema illustrates the link between classes and tables to store objects in a relational database:



Defining data sources in an OOM

You create a data source to define where (in which database) data should be managed by OOM objects.

When you define a data source, you have to declare physical data models in the list of data source models. A data source can contain several models, you can select the source models among a list of models opened in the current workspace. All the selected models represent the same database that will store persistent objects from the OOM.

An OOM can contain several data sources.

Data source properties in an OOM

A data source, includes the following properties:

Property	Description	Maximum length
Name	Name for the data source	254
Code	Reference code for the data source	254
Comment	Descriptive label for the data source	—

A data source also includes the following properties:

Models

The Models page lets you define the list of models representing one database. This database is the source and store of data for the OOM where you define a data source. You use the Add Models tools to open a selection dialog box, you must select among models opened in the current workspace.

Database connection

The Database Connection page lets you define the following parameters to connect to a database:

Parameter	Description
Using an ODBC data source	ODBC data source associated with the database
Using a connection string	Allows to connect to a database via other means than ODBC. A connection string is a series of semicolon delimited arguments that define parameters such as the data source provider and the location of the data source. For example, the connection string for jConnect is: `jdbc:sybase:Tds:machine-name:port-number`
User ID	User login
Password	User password

Creating a data source in an OOM

You can create as many data sources as you need in an OOM. Each data source defines the list of models used as source of data in the current model.

❖ **To create a data source in an OOM:**

1 Select Model→Data Sources to display the List of Data Sources.

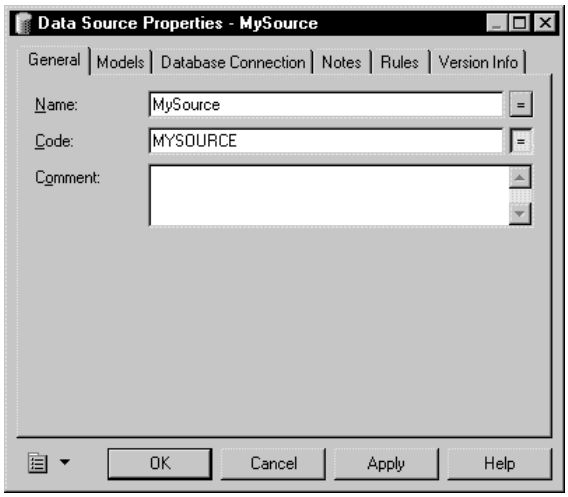
2 Click a blank line in the list.
or
Click the Add a Row tool.

An arrow appears at the beginning of the line.

3 Type a name and a code.

4 Click Apply.

5 Click the Properties button to display the property sheet of the data source.

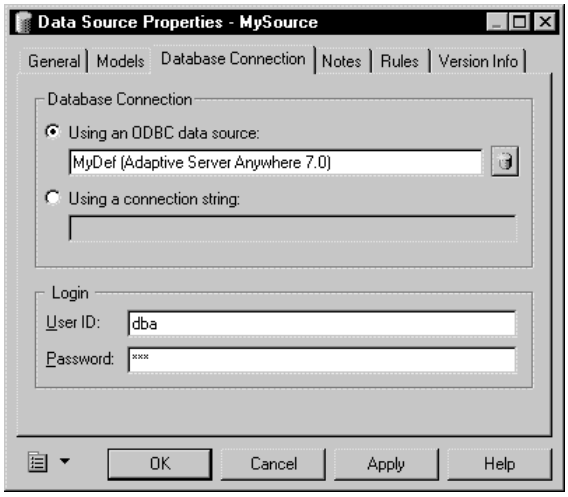


6 Click the Models tab.

7 Click the Add Models tool.

A model selection dialog box appears. You can select physical models for the current data source.

8 Click the Database Connection tab and define the database connection parameters.



9 Click OK in each of the dialog boxes.

Defining O/R mapping

Object persistence requires to store and extract objects in a relational database. You can map OOM objects to PDM objects to create an object to relational mapping:

OOM object	Mapped to…
Class	Table
Attribute	Column
Operation	SELECT or UPDATE queries associated with a table
Association	Reference, view reference, table or view

There are different methods for creating an O/R mapping:

- Automatic mapping
- User-defined mapping

Automatic creation of O/R mapping

When you generate a PDM from an OOM or an OOM from a PDM, and provided you have selected the O/R mapping check box in the generation options, the OOM objects are automatically mapped to the PDM objects and a data source is created in the OOM.

This default mapping can be very useful in the following situations:

- You have designed an OOM and need to define the PDM that will store OOM objects in a relational database. You can generate a new PDM from the current OOM to have OOM objects linked to PDM objects
- You have created a PDM and want to design the corresponding application software in an OOM. You have to generate a new OOM from the current PDM

For more information on OOM to PDM generation, see section Generating a PDM from an OOM, in chapter Generating from an Object-Oriented Model.

For more information on PDM to OOM generation, see section Generating an OOM from a PDM, in chapter Generating from a Physical Data Model in the *PDM User's Guide*.

After generating automatic object mapping, you can fine tune the links between OOM and PDM objects using the user-defined mapping.

User-defined O/R mapping

If your OOM and PDM are already designed, you cannot use the inter-model generation to create automatic mapping between OOM and PDM objects. You then have to define object links manually. The process is time consuming but it provides great flexibility for customizing the mapping between objects and guarantee a correct storage of objects in a relational database.

For more information on user-defined O/R mapping, see section Understanding O/R mapping.

Mapping a class

The following sections explain how to define a mapping for a class and its attributes.

Defining a class mapping

The class mapping allows to identify the physical data objects (tables or views) used to store class instances in a relational database. Once the source tables or views are identified and mapped with the classes of the current model, you can define mappings between attributes and table columns.

You have to use the Mapping page of a class property sheet to define class mappings.

You cannot define any mapping for a class, if you do not select a data source from the Mapping page of a class property sheet. When you declare a mapping between a data source and a class, you indicate the models where objects will be stored.

Data source selection

Tool	Description
	Adds a mapping between the current object and an existing data source. The first time you define a mapping for an object, the Mapping For dropdown listbox is empty, you have to add a mapping to a data source before you can map objects from the current model to objects in the data source
	Deletes the mapping for the data source
	Edits properties of the selected data source

Class sources page

The Class Sources page allows to associate one or several tables or views in the data source to the current class. These tables and views will store the classes in the relational database.

You can use the Add Objects tool to select tables or views from the physical data models opened in the current workspace.

Attributes Mapping page

Each attribute can be associated with a column of the selected tables or views in order to specify where and how attribute values are stored in the database. When a class inherits from a non-generated class via a generalization link, the attributes of the parent class appear in the Selection dialog box in order to let you create a mapping with these inherited attributes. The attributes of a non-generated derived class also appear in the Selection dialog box. You can use the following tools to define attribute mapping.

Icon	Tool	Description
	Add Mapping	Allows you to select the attributes in the current class that will be mapped to columns in the source table or view. When the current class inherits from a non-generated parent or is derived by a non-generated class, the attributes of the parent or derived class appear in the selection dialog box. Once you have selected the attributes, you can use the dropdown listbox in the Mapped to column to select corresponding columns in the source table or view
	Create from Sources	Allows you to copy columns from the tables or views in the data source to the current class. The name, code , description, annotation, and comment are copied and the data types are converted in order to match the current model
	Generate Mapping	Allows you to automatically generate a mapping between attributes and columns with same name or code in the data source and the current model

The Ellipsis button in the Mapped To column opens the SQL Editor dialog box to let you customize the source expression for the attribute.

Operations Mapping page

The Operations Mapping page allows to associate a SQL query to an operation of a class.

For more information on operation mappings, see section Mapping an operation.

Criteria page

The Criteria page allows to specify join criteria between source tables or views. For example: **`EMPLOYEE.ID < 100`**

SQL queries

The following SQL queries are automatically computed to figure out how data of class instances are retrieved from or stored into the database:

- The **Select** page displays a statement that allows to retrieve attribute values of class instances from the database using the class identifying attributes. For example:

```
select
      ADDRESS.LINE1 "LINE1",
      ADDRESS.LINE2 "LINE2",
      ADDRESS.CITY "CITY",
      ADDRESS.ZIPCODE "ZIPCODE",
      ADDRESS.COUNTRY "COUNTRY"
from  ADDRESS
```

- The **Insert** page displays a statement that allows to create an instance of the class and save it with its attributes. For example:

```
insert into ADDRESS(
     ADDRESS.LINE1,
     ADDRESS.LINE2,
     ADDRESS.CITY,
     ADDRESS.ZIPCODE,
     ADDRESS.COUNTRY)
values (
     %LINE1%,
     %LINE2%,
     %CITY%,
     %ZIPCODE%,
     %COUNTRY%)
```

- The **Update** page displays a statement that allows to update attribute values of class instances apart from identifying attributes. For example:

```
update ADDRESS
set   ADDRESS.LINE1 = %LINE1%,
      ADDRESS.LINE2 = %LINE2%,
      ADDRESS.CITY = %CITY%,
      ADDRESS.ZIPCODE = %ZIPCODE%,
      ADDRESS.COUNTRY = %COUNTRY%
```

- The **Delete** page displays a statement that allows to delete a class instance from the database using its identifying attributes. For example:

```
delete line* from ADDRESS
```

Each statement is automatically generated. If you modify a statement, it becomes user-defined and can no longer be automatically calculated, even when you update the mapping. You can tell that a statement is user-defined because the User-Defined tool is pressed in the toolbar.

You can recover the computed expression by clicking the User-Defined tool, this will remove any non calculated expression from the page.

Modify query template

The syntax of SQL queries follows a predefined template. You can customize the query syntax by modifying the **SelectStatement**, **InsertStatement**, **UpdateStatement**, and **DeleteStatement** entries in the Profile category of the object language or extended model definition.

For more information on customizing templates, see section Defining templates, in chapter Generation Reference Guide in the *Advanced User Documentation*.

Creating an object to relational mapping for a class

❖ **To create an object to relational mapping for a class:**

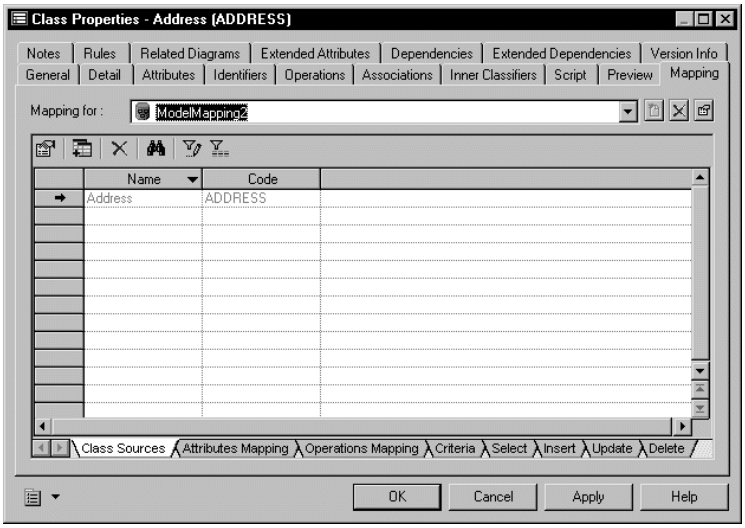
1 Double-click a class to display the class property sheet.

2 Click the Mapping tab to display the Mapping page.

3 Click the Add A Mapping For A Data Source tool to select a data source for the mapping.
or
Select a data source in the Mapping For dropdown listbox if you have already selected one or several data sources.

4 In the Class Sources page, click the Add Objects tool to select the source tables and views you want to map to the current class.

The name and code of the tables and views appear in the list.



5 Click the Attributes Mapping tab in the Mapping page.

6 Click the Add Mapping tool.

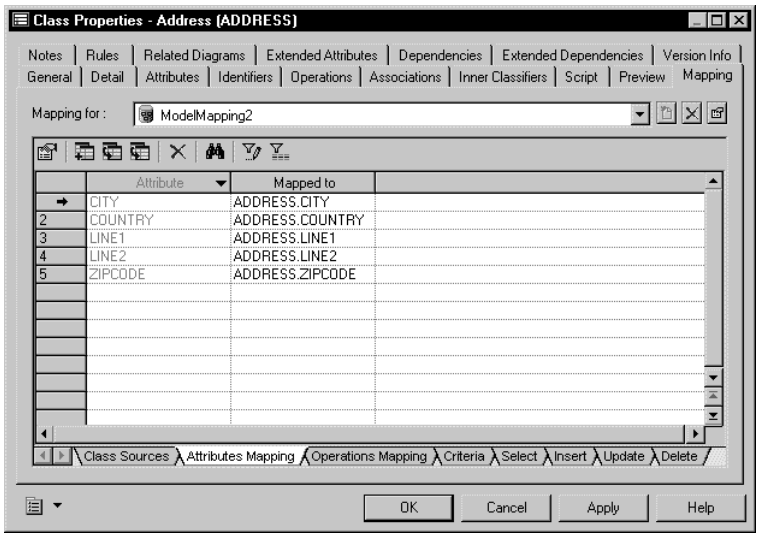
7 Select the attributes that will be mapped to columns in the source tables or views and click OK.

The selected columns appear in the first column of the list.

> **Generate mapping**
> You can click the Generate Mapping tool to automatically create the mapping between columns and attributes with identical name or code in the data source and the current model

8 Click the down arrow in the Mapped To column to select the column in the data source table to which the selected attribute will be mapped.

You have to repeat this operation for each attribute you want to map to the data source.



9 Click the Criteria tab if you want to specify join criteria between the source tables or views.

10 Select the different queries tabs to visualize or modify the SQL statements.

11 Click OK.

Mapping an operation

When an operation implies action on the database, like data retrieval or data update, you can associate this operation with a SQL query that will execute an action in the database. Usually, this query is a SELECT statement used to retrieve one or several instances of the current class according to the operation parameter. It can also be an UPDATE query.

In the context of an EJB, the SQL query mapped to an operation is used to implement finder or select methods.

Defining an operation mapping

Data source selection

You cannot define any mapping for an operation, if you do not select a data source from the Mapping page of an operation property sheet.

For more information on data source selection, see section Defining a class mapping

Criteria page

The Criteria page allows to specify join criteria between source tables or views. For example: **`EMPLOYEE.ID < 100`**

Query

This query is automatically computed to figure out how to retrieve or update data in the database. If you modify the SQL query, it becomes user-defined and can no longer be automatically calculated, even when you update the mapping. You can tell that a statement is user-defined because the User-Defined tool is pressed in the toolbar.

You can recover the computed expression by clicking the User-Defined tool, this will remove any non calculated expression from the page.

Modify query template

The syntax of the SQL query follows a predefined template. You can customize the query syntax by modifying the **Query** entry in the Profile category of the object language or extended model definition.

For more information on customizing templates, see section Defining templates, in chapter Generation Reference Guide in the *Advanced User Documentation*.

Creating an object to relational mapping for an operation

❖ **To create an object to relational mapping for an operation:**

1 Double-click a class to display the class property sheet.

2 Click the Operations tab to display the Operations page.

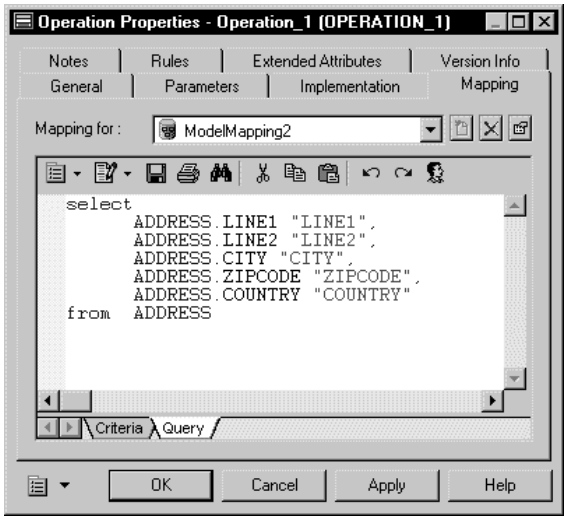
3 Double-click an operation to display the operation property sheet.

4 Click the Mapping tab to display the Mapping page.

5 Click the Add A Mapping For A Data Source tool to select a data source for the mapping.
or
Select a data source in the Mapping For dropdown listbox if you have already selected one or several data sources.

6 Click the Criteria tab if you want to specify join criteria between the source tables or views.

7 Click the Query tab to visualize or modify the query statement.



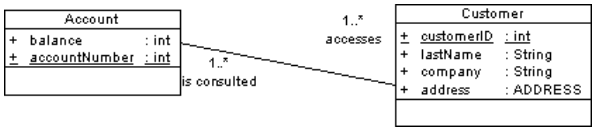
8 Click OK.

Mapping an association in a class diagram

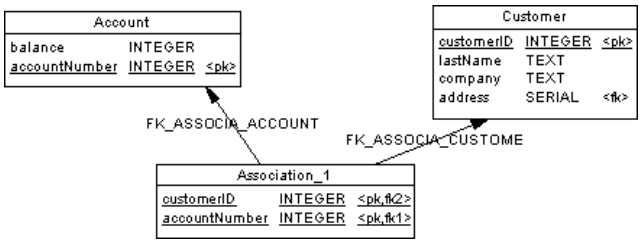
O/R mapping on an association allows to define the **role navigability** of an association in the database. When an association role is not navigable, there is no need to design a mapping for the association since no information is transmitted between the classes. However, when an association role is navigable, you have to set up the structure for data transmission within the database. In relational databases, data transmission is implemented via foreign keys designed to relate a record in one table with a record in another.

The type of mapping for an association depends on the association multiplicity:

- One-to-one or one-to-many associations can be mapped to a reference in the source model. This reference is used in the database to transfer data and migrate key columns to the appropriate table
- Many-to-many associations have to be mapped to an associative table. This table is created to maintain a relationship between two or more tables in a relational database. The columns contained in the associative table are the combination of the keys in the tables involved in the reference. For example, the following classes have a many-to-many association:



In a relational database, this association is mapped to the following associative table:



SQL queries

When the association roles are navigable, the following queries are automatically computed:

Query	Action
Select (Role A)	Retrieves related class A instances for the role A defined on class B
Insert (Role A)	Associates an instance of class A with class B. It is computed only when the source association is an associative table
Delete (Role A)	Deletes from class B the association with class A. This query is computed only when the source association is an associative table
Select (Role B)	Retrieves related class B instances for the role B defined on class A

Query	Action
Insert (Role B)	Associates an instance of class B with class A. This query is computed only when the source association is an associative table
Delete (Role B)	Deletes from class A the association with class B. This query is computed only when the source association is an associative table

Each statement is automatically generated. If you modify a statement, it becomes user-defined and can no longer be automatically calculated, even when you update the mapping. You can tell that a statement is user-defined because the User-Defined tool is pressed in the toolbar. You can recover the computed expression by clicking the User-Defined tool. This removes any non-calculated expression from the page.

Modify query template

The syntax of SQL queries follows a predefined template. You can customize the query syntax by modifying the **SelectA**, **InsertA**, **UpdateA**, **DeleteA**, **SelectB**, **InsertB**, **UpdateB**, and **DeleteB** entries in the Profile category of the object language or extended model definition.

For more information on customizing templates, see section Defining templates, in chapter Generation Reference Guide in the *Advanced User Documentation*.

Creating an object to relational mapping for an association in a class diagram

❖ **To create an object to relational mapping for an association in a class diagram:**

1. Double-click an association to display the association
2. Click the Mapping tab to display the Mapping page.
3. Click the Add A Mapping For A Data Source tool to select a data source for the mapping.
   or
   Select a data source in the Mapping For dropdown listbox if you have already selected one or several data sources.
4. Click the Criteria tab if you want to specify join criteria between the source tables or views.
5. Click a query tab to visualize or modify the query statement.
6. Click OK.

Opening a Rose model in an OOM

You can open a model built with Rational Rose (.MDL) version 98, 2000 and 2002.

A new OOM is created for the Rose model, and the objects of the Rose model are translated into OOM objects.

This functionality provides you with greater scope and flexibility. You can create an OOM from a Rose model, from which you generate Java files or objects for PowerBuilder to create applications. You can also use the OOM created from a Rose model to add to an existing OOM, or to generate a CDM or PDM for database analysis purposes.

Object language of the new OOM

A Rose model can support one or several languages whereas a PowerDesigner OOM has a single object language. When you open a Rose model in an OOM:

If the Rose model supports only one object language the new OOM will have the same object language.

If the Rose model supports several object languages the new OOM will have only one of the object languages of the Rose model.

Java, C++, Visual Basic, XML-DTD and Analysis languages from a Rose model are mapped to the same languages in PowerDesigner.

The following table maps Rose languages to PowerDesigner languages:

Rose language	PowerDesigner language
Oracle 8, Ada, COM, Web Modeler	Analysis
VC++	C++

❖ **To open a Rose model in an OOM:**

1 Select File→Import→Rational Rose File to display the open file dialog box.

2 Select or browse to the directory that contains the Rose file.

3 Select Rational Rose Model (*.MDL) file from the Files of Type dropdown listbox.

4 Select a file.

5 Click Open.

The Output window displays a message following the successful import and the default diagram of the model appears.

Importing general objects

When you open a Rose model in an OOM, the following objects are directly imported:

Package
Diagram
Note
Note Link
Text

Property in a Rose model	Becomes in an OOM
Documentation	Comment
Zoom	Page scale
Stereotype	Stereotype

The following properties are **not** imported into the new OOM:

Package

Global

Importing use case diagrams

The import process translates some properties of imported objects into OOM properties in the following way:

What is imported

The following properties are imported into the new OOM:

Actor

Property in a Rose model	Becomes in an OOM
Class with <<actor>> stereotype	Implementation class

Use case association

Property in a Rose model	Becomes in an OOM
Diagrams	Related diagrams

What is not imported

The following properties are **not** imported into the new OOM:

Actor	Actors in Rose are classes with an <<actor>>, <<business actor>>, or <<business worker>> stereotype, whereas in an OOM, actors are specific objects. When you import a Rose model containing actors, PowerDesigner creates: ♦ A class to recover the operations and attributes of the Rose actor. This class has the same stereotype as in Rose, it will be used as implementation class for the new actor. ♦ An actor with an implementation class
Use case	Rank Abstract
Use case association	Associations between two actors or two use cases in the Rose model are not supported in an OOM. Role name for class A, for class B Export control for class A, for class B Cardinality for class A, for class B Aggregate for class A, for class B Aggregate for class A, for class B (mutually exclusive) By reference By value Unspecified Documentation for class A, for class B Keys / qualifiers for class A, for class B Constraints for class A, for class B Constraints Derived Static Friend
Use case dependency	Dependencies between a use case and an actor in the Rose model are not supported in an OOM.
Class	Rose classes with the following stereotypes are imported in an OOM, but the symbol of the classes are not imported because the use case diagram does not contain any classes: Boundary Business entity Control Entity Table

Importing class diagrams

The import process translates some properties of imported objects into OOM properties in the following way:

What is imported

The following properties are imported into the new OOM:

Class

Property in a Rose model	Becomes in an OOM
Class utility (Type)	Utility (Type)
Export Control	Visibility
Public	Public
Protected	Protected
Private	Private
Implementation	Package
Cardinality: 0..n, 1..n	Cardinality: 0..*, 1..*
Nested class	Inner classifier

Interface

Property in a Rose model	Becomes in an OOM
Export Control	Visibility
Public	Public
Protected	Protected
Private	Private
Implementation	Package
Nested class	Inner classifier

Attribute

Property in a Rose model	Becomes in an OOM
Export Control	Visibility
Public	Public
Protected	Protected
Private	Private
Implementation	Package

Operation

Property in a Rose model	Becomes in an OOM
Export Control	Visibility
Public	Public
Protected	Protected
Private	Private
Implementation	Package

Generalization

Property in a Rose model	Becomes in an OOM
Export Control	Visibility
Public	Public
Protected	Protected
Private	Private
Implementation	Package
Virtual inheritance	Extended attribute

Association

Property in a Rose model	Becomes in an OOM
Export Control	Visibility
Cardinality	Multiplicity
Aggregate	Aggregation/Composition
Aggregate (class A or B)	Container
By reference	Aggregation
By value	Composition
Unspecified	No aggregation or composition

What is not imported

The following properties are **not** imported into the new OOM:

Class

Type
- Parameterized Class
- Instantiated Class
- Parameterized Class Utility
- Instantiated Class Utility
- MetaClass

Formal arguments

Concurrency
- Sequential
- Guarded
- Active
- Synchronous

Interface

Type
- Class
- Class Utility
- Parameterized Class
- Instantiated Class
- Parameterized Class Utility
- Instantiated Class Utility
- MetaClass

	Formal arguments Cardinality Persistence Abstract Concurrency Sequential Guarded Active Synchronous
Attribute	Containment By Value By Reference Unspecified
Operation	Arguments Name Type Default Protocol Qualification (language-specific) Exceptions Size (amount of storage) Time (to complete operation) Concurrency (sequential, guarded, synchronous) Preconditions Postconditions
Generalization	FriendshipRequired (Yes/No)
Association	Documentation for class A, for class B Keys/qualifiers for class A, for class B Constraints for class A, for class B Constraints Stereotype Derived Static Friend
Dependency	ExportControl FriendshipRequired Cardinality from Cardinality to

Importing collaboration diagrams

The following table maps Rose objects to PowerDesigner objects in the following way:

Rose object name	PowerDesigner object name
Object & Class Instance	Object
Link or Object Link	Instance link

What is imported

The following properties are imported into the new OOM:

Object

Property in a Rose model	Becomes in an OOM
Class	Class
Multiple instances	Multiple

Instance link

Property in a Rose model	Becomes in an OOM
Assoc	Association
Messages list	Messages

Message

Synchronization.

The following table maps message stereotypes in Rose and the corresponding control flow types in PowerDesigner:

Rose Synchronization	PowerDesigner Control Flow type
Simple	Undefined
Synchronous	Procedure call
Asynchronous	Asynchronous
Balking	Imported as a message condition
Time out	Not imported

- Balking: the client passes the message only if the supplier is immediately ready to accept the message: it is imported as a message condition.
- Timeout: if the message is not acknowledged within a timeout, the message is abandoned: this property is not imported.

What is not imported

The following properties are **not** imported into the new OOM:

Object

Persistence (persistent, static, transient)

Instance link

Shared/Unshared check box
Client visibility: unspecified, field, parameters, local, global
Supplier visibility: unspecified, field, parameters, local, global

Package

In Rose, a package cannot be included into a collaboration diagram.

Importing sequence diagrams

The import process translates some properties of imported objects into OOM properties in the following way:

What is imported

The following properties are imported into the new OOM:

Message

Property in a Rose model	Becomes in an OOM
Return message	Message with its control flow set to Return
Destruction marker symbol	Recursive message with its action set to Destroy (imported for objects, not for actors)

Synchronization.

The following table maps message stereotypes in Rose and the corresponding control flow types in PowerDesigner:

Rose Synchronization	PowerDesigner Control Flow type
Simple	Undefined
Procedure call	Procedure call
Asynchronous	Asynchronous
Balking	The client passes the message only if the supplier is immediately ready to accept the message: it is imported as a message condition
Time out	If the message is not acknowledged within a timeout, the message is abandoned and this property is not imported

What is not imported

The following properties are **not** imported into the new OOM:

Object	Multiple instances
Message	Frequency (Aperiodic, Periodic) Synchronization (synchronous)

Importing statechart diagrams

The following table maps Rose objects to PowerDesigner objects in the following way:

Rose object name	PowerDesigner object name
State containing symbols in at least one statechart diagram	State
State containing symbols in at least one activity diagram and only if it is related to an Object	Object state (in an activity diagram)
State transition	Transition

In Rose, the activity and statechart diagrams are created in the Use Case View or in the Logical View:

- At the root level
- In an activity
- In a state

A UML State Machine is automatically created: it contains statechart and also activity diagrams with their relevant objects. In PowerDesigner, statechart diagrams are created at the model level or in a composite state: the parent package or the model is considered the State Machine.

What is imported

The following properties are imported into the new OOM:

Statechart diagram	Rose statechart diagrams that are either at the root level, or in a state are imported. Rose statechart diagrams that are in an activity are not imported: an error message is displayed in the Output window.

State

Property in a Rose model	Becomes in an OOM
Actions	Actions

Transition

Property in a Rose model	Becomes in an OOM
<No name>	Calculated name
<No code>	Calculated code
Event	Trigger event
Arguments	Event arguments
Guard condition	Condition
Action	Trigger action

Action

Property in a Rose model	Becomes in an OOM
When	Trigger event
OnEntry	Entry
OnExit	Exit
Do	Do
OnEvent	Event (or reuses an existing event with the same name)
Event	Trigger event
Event arguments	Event arguments

What is not imported

The following properties are **not** imported into the new OOM:

Statechart diagram

Rose statechart diagrams that are in an activity are not imported: an error message is displayed in the Output window.

State

State & Sub state
    Activity history

Action

Event condition
Type
    Send Event

Start & End

Actions

Transitions
Swimlanes

Transition

Transitions between two Sub states
Event
    Send Events
Arguments
    Send Arguments
Action
    Send Target

Package

In Rose, a package cannot be included into a statechart diagram.

Importing activity diagrams

The following table maps Rose objects to PowerDesigner objects in the following way:

Rose object name	PowerDesigner object name
Activity	Activity
StateMachine	—
Object (associated with a State)	Object state
State	State (imported in the model, but there is no symbol in the activity diagram because a state is only displayed in a statechart diagram)
Start state	Start
Self Transition & Object Flow	Transition
Synchronization	Synchronization
Decision	Decision
End state	End
Swimlane	Organization unit/swimlane

What is imported

The following properties are imported into the new OOM:

Activity

Property in a Rose model	Becomes in an OOM
Actions	Action

Rose allows multiple actions, PowerDesigner allows only one action. After import in an OOM, the Action page in the activity property sheet displays <<Undefined>> and the text editor reproduces the list of actions coming from the action property page in Rose.

Transition

Property in a Rose model	Becomes in an OOM
Guard condition	Condition

About Rose Subunits

PowerDesigner does not manage Rose Subunits as separate files, it imports *.CAT and *.SUB files in the same model that refers to them. If a .CAT or a .SUB file does not exist in the specified path, PowerDesigner searches in the same directory as the file containing the model.

What is not imported

The following properties are **not** imported into the new OOM:

Activity

State & Sub state
    Activity history
Transitions
    State
Swimlanes

Object state

In Rose, you can associate an Object (instance of a class) with a State.

In an OOM, the Rose Object is imported as an object without symbol. If the Rose Object is associated with a State, an object state with a symbol is created in the OOM and contains the name, stereotype and comment of the State.

If the Rose diagram that contains the symbol of the Object is in a composite activity, PowerDesigner creates a shortcut of the object in the composite activity in the OOM. This happens because an object state cannot be decomposed in an OOM.

Start & End

Actions
Transitions
Swimlanes

Transition

Transitions between two states
Event
    Send Events
Arguments
    Send Arguments
Action

Send Target
Name (in PowerDesigner, the name of the transition is calculated after the source and destination objects names).

Decision

Swimlanes

Importing component diagrams

The import process translates some properties of imported objects into OOM properties in the following way:

What is imported

The following properties are imported into the new OOM:

Component

Property in a Rose model	Becomes in an OOM
Interfaces of Realizes	Interfaces
Classes of Realizes	Classes
Files	External files in Extended Dependencies
URL files	URL files in Extended Dependencies
Declarations	Description in the Notes page

The following types of components have Rose predefined stereotypes with different symbols:

Active X
Applet
Application
DLL
EXE
Generic Package
Generic Subprogram
Main Program
Package Body
Package Specification
Subprogram Body
Subprogram Specification
Task Body
Task Specification

In PowerDesigner, these stereotypes are imported with the same symbol.

What is not imported

The following property is **not** imported into the new OOM:

Component

Language

The PowerDesigner language is only a property of the model, not of the component. By convention, the language property is initialized with the Rose Language property of the first object found in the Rose model.

Importing deployment diagrams

The import process translates some properties of imported objects into OOM properties in the following way:

What is imported

The following properties are imported into the new OOM:

Node

Property in a Rose model	Becomes in an OOM
Device	Node
Processor	Node
Device characteristics	Description
Processor characteristics	Description

File objects

Property in a Rose model	Becomes in an OOM
File	File object linked to the package
URL definition	File object

Node association

Property in a Rose model	Becomes in an OOM
Connection	Node association
Characteristics	Description

What is not imported

The following property is **not** imported into the new OOM:

Node

Process objects are not imported: a warning message is displayed in the Output window.

	Processor Processes list (priority), Scheduling (preemptive,...)
Package	In Rose, a package cannot be included into a deployment diagram.

CHAPTER 14

Reverse Engineering

About this chapter

This chapter describes Java, XML, and other reverse engineering functions for an Object-Oriented Model (OOM), it also describes EJB reverse engineering. It shows you how to reverse engineer into an OOM and at the end, how to create a new OOM using reverse engineering.

Contents

Topic	Page
What is reverse engineering?	590
Reverse engineering Java	591
Reverse engineering PowerBuilder	607
Reverse engineering XML	613
Reverse engineering into a new OOM	617

What is reverse engineering?

Reverse engineering is the process of examining and recovering data or source code from a file that is then used to build or update an OOM. You reverse engineer objects to an OOM via a diagram.

You can reverse engineer objects to a new model, or to an existing model. When you reverse engineer an object which already exists in a model, you use an object comparison box to choose either to replace the existing object, or to keep the existing object in the model.

> **Parsing**
> PowerDesigner uses a parser software for XML reverse engineering, developed by the Apache Software Foundation (http://www.apache.org/).

You can reverse the following types of files into an OOM:

- Java
- PowerBuilder
- XML
- J2EE components

Object Language reverse engineering

If you use reverse engineering into an OOM, you can easily access the reverse engineering options of an object language that is not the current object language. This functionality is accessible from File→Reverse Engineer→Object Language.

Inner classifiers

When you reverse a language containing one or more inner classifiers into an OOM, one class is created for the outer class, and one class is created for each of the inner classifiers. Moreover, an inner link is created between each inner classifier and the outer class to which it belongs. Inner classifiers are commonly used in Java, and XML.

For more information on inner classifiers, see section Defining inner classifiers, in chapter Building a class diagram.

Symbol creation

If you select the Create Symbols reverse option, the layout of the symbols in the diagram is automatically arranged. In case of a reverse engineering with a large number of objects with complex interactions, the auto-layout feature is likely to create synonyms of objects to improve the diagram readability. For example, if a class has a large number of associations, the auto-layout feature will create a synonym of this class in another location of the diagram in order to improve the diagram presentation.

Reverse engineering Java

You can reverse engineer files that contain Java classes into an OOM.

For each existing class in a Java file, a corresponding class is created in the OOM, with the same name and containing the same information. When you reverse engineer a Java class that already exists in a model, you can choose in the Merge Model window, either to replace the existing class, or to keep the existing class definition in the model.

Reverse engineered Java classes always keep their original names.

Defining public class

Java allows the creation of multiples classes in a single file but one class, and only one, has to be public. While reversing, a dependency link with a specific stereotype <sameFile> is created between the public class and the dependent class. The Java generation process handles this link during generation as well.

Reverse engineering Java formats

You reverse engineer classes from Java files to a class diagram from Language→Reverse Engineer Java.

You can choose one of the following Java formats from the Reverse dropdown listbox:

Format	Description	Extension
Java files	File containing one or several class definitions	.java
Java directories	Folder from which you can reverse all .java files, including those contained in its sub-directories	—
Class files	File containing one class definition with the same name as the file	.class
Class directories	Folder from which you can reverse all .class files, including those contained in its sub-directories	—
Archive	Compressed .jar or .zip files. Only the Java .class files contained in these files are imported into your model. All other information is discarded	.zip and .jar

You cannot select more than one Java format to reverse at the same time.

File encoding

If you reverse from Java and you know which encoding format is used within the sources, you can select the appropriate encoding format from the Selection page, otherwise your can use the ANSI (Auto Detection) value which tries to auto-detect the encoding style and uses ANSI if it fails to detect.

Under Windows 98 and Millenium, PowerDesigner cannot benefit from the auto-detect MS method, therefore you should select the encoding style you need rather than let the auto-detect feature try to detect the encoding style.

Java code comments

When you reverse engineer Java files, some comments may change form or position within the code.

Comment in original Java file	After reverse
Before the import declarations	Goes to header
Beginning with /*	Begins with //
At the end of the file below all the code	Goes to footer
Within a class but not within an operation	Is attached to the attribute or operation that immediately follows it

Reverse engineering Java options

You define Java reverse engineering options from the Reverse Java dialog box from Language→Reverse Engineer Java.

You can define the following Java reverse engineering options:

Option	Result of selection
Ignore operation body	Reverses classes without including the body of the code
Ignore comments	Reverses classes without including code comments
Create symbols	Creates a symbol for each object in the diagram, if not, reversed objects are only visible in the browser
Mark classifiers not to be generated	Reversed classifiers (classes and interfaces) will not be generated from the model. To generate the classifier, you must select the Generate check box in its property sheet
Create Associations	Creates associations between classes and/or interfaces
Reverse engineer Deployment Descriptor	Reverses components with deployment descriptor

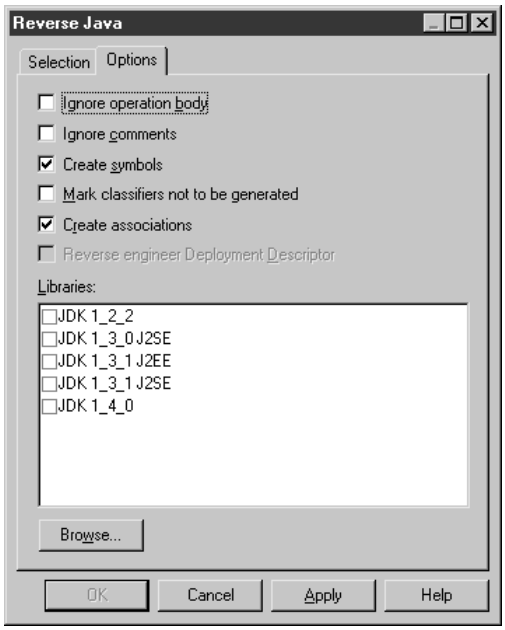
Option	Result of selection
Libraries	Opens a JDK model in the workspace. The Setup program installs these models that contain the class libraries of each version of JDK. You can load them quickly into PowerDesigner and save time reversing them

For more information on the Reverse engineer Deployment Descriptor option, see the different Reverse engineering sections, in chapter Working with platform dependent components.

❖ **To define Java reverse engineering options:**

1 Select Language→Reverse Engineer Java to display the Reverse Java dialog box.

2 Click the Options tab to display the Options page.



3 Select or clear options.

4 Click Apply and Cancel.

Reverse engineering Java files

You can reverse engineer different types of Java files into an OOM.

Reverse engineering Java files without code body

You can reverse engineer .java class files without the body of the code. When you reverse classes this way, the code contained within the operations of the class does not appear in the reversed class.

You use this option when you want to reverse objects for visualization or comparison purposes, or to limit the size of your model if you have a very large number of classes to reverse.

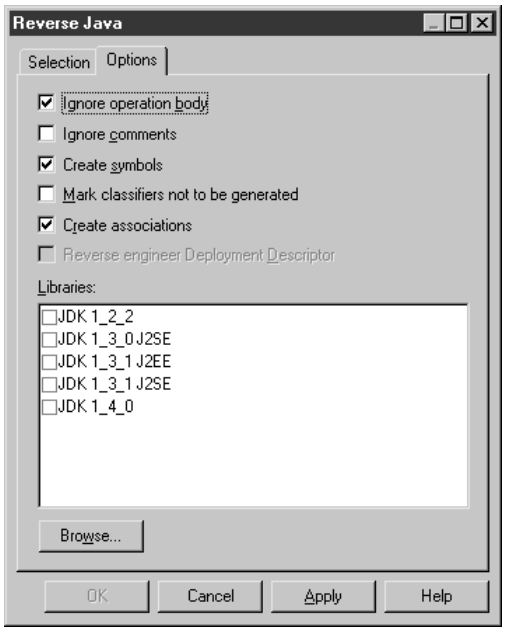
❖ **To reverse java without code body:**

1. Select Language→Reverse Engineer Java. to display the Reverse Java dialog box.
2. Select Java files from the Reverse dropdown listbox in the Selection page.
3. Select another file encoding type from the File Encoding dropdown listbox if needed.
4. Click the Add button.

   A standard Open dialog box appears.
5. Select the .java files you want to reverse and click Open.

   The Reverse Java dialog box displays the files selected.
6. Click the Options tab to display the Options page.

7 Select the Ignore operation body check box.



8 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

9 Click OK.

The classes are reversed without code body. They are automatically added to your model and visible in the diagram. They are also listed in the Reverse page of the Output window, located in the lower part of the main window.

Reverse engineering .java files

Each .java file contains information on one or several class definitions. When you reverse engineer a .java file, PowerDesigner creates a class in the model corresponding to each class definition in the .java file. The newly created classes have the same name as in the .java file.

❖ **To reverse engineer .java files:**

1 Select Language→Reverse Engineer Java to display the Reverse Java dialog box.

2 Select Java files from the Reverse dropdown listbox in the Selection page.

> **Reversing without the body of the code**
> You can choose to reverse .java source files without code body by selecting the Ignore operation body check box in the Options page.

3 Select another file encoding type from the File Encoding dropdown listbox if needed.

4 Click the Add button.

A standard Open dialog box appears.

5 Select the .java files you want to reverse and click Open.

The Reverse Java dialog box displays the files selected.

6 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

7 Click OK.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

The classes are added to your model. They are visible in the diagram and in the Browser. They are also listed in the Reverse page of the Output window, located in the lower part of the main window.

Reverse engineering compiled Java files

A compiled .class file contains the definition of one unique class. A compiled .class file results from compiling a .java file using an independent Java compiler. After compilation, each class definition in the .java file becomes an individual compiled .class file.

When you reverse engineer a .class file, PowerDesigner creates a class in the model that corresponds to the class definition in the .class file. The newly created class has the same name as the definition in the .class file.

❖ **To reverse engineer compiled Java files to a diagram:**

1 Select Language→Reverse Engineer Java to display the Reverse Java dialog box.

2 Select Class files from the Reverse dropdown listbox in the Selection page.

3 Select another file encoding type from the File Encoding dropdown listbox if needed.

4 Click the Add button.

A standard Open dialog box appears.

5 Select the .class files you want to reverse and click Open.

> **Multi-selection**
> You select several files simultaneously using the CTRL or SHIFT keys.

The Reverse Java dialog box displays the files selected.

6 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

7 Click OK.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

The classes are added to your model. They are visible in the diagram and in the Browser. They are also listed in the Reverse page of the Output window, located in the lower part of the main window.

Reverse engineering Java or Class files from a directory

You follow the same procedure when reverse engineering .java or .class files from a source directory as when you reverse engineer independent .java or .class files, the only difference is that you select a directory in which several .java or .class files are located. You can thus reverse groups of files that belong to the same model or package.

Java library files are often interdependent as they belong to the same model and are therefore located in the same directory. In this case, if you do not reverse engineer all the library files located in the directory, your model may be incomplete.

When you reverse engineer a directory, all the sub-directories and the Java files contained in them are reversed. As a result, each sub-directory becomes a package within the model.

❖ **To reverse engineer Java or Class files from a directory:**

1 Select Language→Reverse Engineer Java to display the Reverse Java dialog box.

2 Select Java directories or Class directories from the Reverse dropdown listbox in the Selection page.

> **Reversing without the body of the code**
> You can choose to reverse .java source files without code body by deselecting the Ignore operation body check box in the Options page.

3 Select another file encoding type from the File Encoding dropdown listbox if needed.

4 Click the Add button.

The Browse for Folder dialog box appears.

5 Select the directory containing the Java or Class files to reverse and click OK.

The Reverse Java dialog box displays the directory selected.

6 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

The classes are added to your model. They are visible in the diagram and in the Browser. They are also listed in the Reverse page of the Output window, located in the lower part of the main window.

Reverse engineering archived .jar or .zip files

Each .jar or .zip file contains definitions of one or several classes. These files can often contain large numbers of class definitions.

A .jar file is a compressed file type that contains one or several Java .class definitions. When you reverse engineer a .jar file, PowerDesigner creates a class for each class definition in the .jar file.

A .zip file can contain one or several Java .class files, as well as other files. When you reverse engineer a .zip file, only the Java .class files are reversed. Information stored in other files is not reversed and is totally discarded.

❖ **To reverse engineer .jar or .zip files to a diagram:**

1 Select Language→Reverse Engineer Java to display the Reverse Java dialog box.

2 Select Archive from the Reverse dropdown listbox in the Selection page.

3 Select another file encoding type from the File Encoding dropdown listbox if needed.

4 Click the Add button.

A standard Open dialog box appears.

5 Select the .jar or .zip containing the .class files to reverse and click Open.

The Reverse Java dialog box displays the files you selected.

6 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

The classes are added to your model. They are visible in the diagram and in the Browser. They are also listed in the Reverse page of the Output window, located in the lower part of the main window.

Loading a JDK library model in the workspace

The Java Development Kit (JDK) contains the software and tools that developers need to compile, debug, and run applets and applications written in the Java programming language.

When you reverse engineer Java files, you can at the same time, load one of the JDK models that contains the class libraries of a particular version of JDK. The Setup program installs these models in the Library directory. You can select to reverse a JDK library model from the Options page of the Reverse Java dialog box.

You can open a JDK library model in the workspace from the Library directory, and then reference a class from the reversed JDK library model by creating shortcuts.

❖ **To load a JDK library model:**

1 Select File→Open to display the Open dialog box.

2 Select or browse to the Library directory.

The available library files are listed. Each JDK file corresponds to a particular version of JDK.

3 Select the file JDK-1_4_0.OOM

This file contains all the library class files of version 1_4_0 of JDK.

4 Click Open.

The OOM opens in the workspace.

Reverse engineering comments using Javadoc

You recover Javadoc comments in classes, interfaces, operations, and attributes during reverse engineering. Javadoc comments are reverse engineered only if they exist in the source code.

The Reverse Javadoc Comments feature is very useful for **round-trip engineering**: you keep Javadoc comments during reverse engineering and you can regenerate the code with preserved Javadoc comments.

For more information on the generation of Javadoc comments, see section Generating comments using Javadoc, in chapter Generating for a language.

Using Javadoc comments

At any time, you can view the HTML pages corresponding to the generated or reverse engineered Javadoc comments.

The following table maps Javadoc comments with object properties in PowerDesigner:

Javadoc comment	For	Corresponds to…
%comment%	Class, interface, operation, attribute	Comment box. If Javadoc comments are not found, standard comments are reversed instead
@author	Class, interface	If the tag is not defined, the user name from the Version Info page is used, otherwise the defined tag value appears
@since	Class, interface, operation, attribute	Javadoc@since extended attribute
@deprecated	Class, interface, operation, attribute	Javadoc@deprecated extended attribute
@version	Class, interface	Javadoc@version extended attribute
@see	Class, interface, operation, attribute	Javadoc@see extended attribute
@return	Operation	Javadoc@return extended attribute
@exception	Operation	Javadoc@exception extended attribute. You can declare operation exceptions
@throws	Operation	Javadoc@exception extended attribute. You can declare operation exceptions
@serialData	Operation	Javadoc@serialData extended attribute
@serialField	Attribute	Javadoc@serialField extended attribute
@serial	Attribute	Javadoc@serial extended attribute

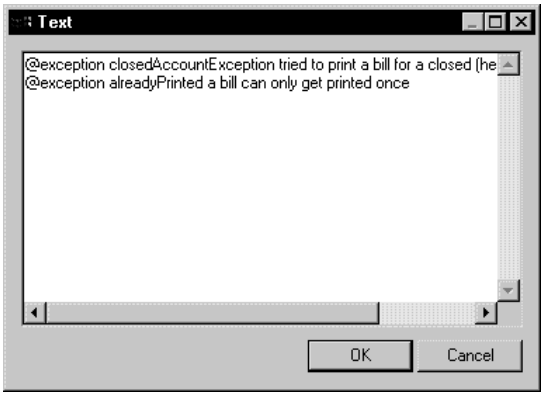
Defining values for Javadoc tags

You can define values for Javadoc tags from the Extended Attributes page of an object property sheet.

To do so, you have to select a Javadoc tag in the list of extended attributes and click the ellipsis button in the Value column. The input dialog box that appears allows you to create values for the selected Javadoc tag. For example, if the data type is set to (Color), you can select another color in the Color dialog box by clicking the ellipsis button in the Value column.

Javadoc tags are not generated if no extended attribute is valued.

Do not forget to repeat the Javadoc tag before each new value, some values being multi-line. If you do not repeat the Javadoc tag, the values will not be generated.


Text
@exception closedAccountException tried to print a bill for a closed (he
@exception alreadyPrinted a bill can only get printed once
OK
Cancel

When you assign a value to a Javadoc tag, the tag and its value appear in the code preview page of the corresponding object.

@author

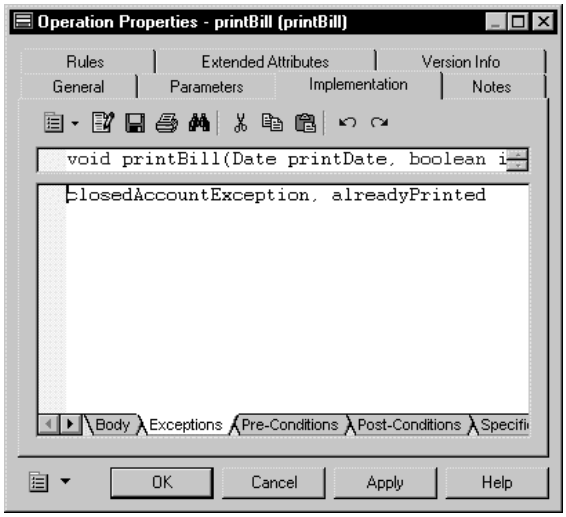
@author is not generated if the extended attribute has no value.

@exceptions and @throws

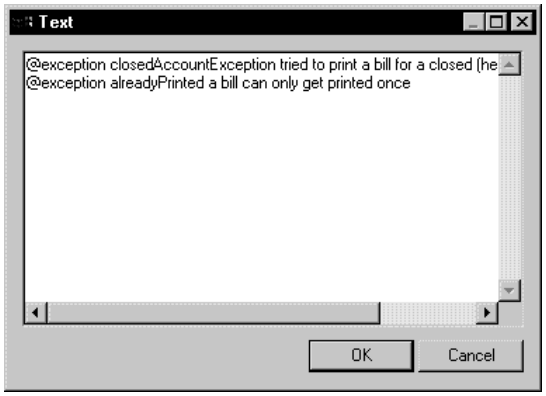
@exceptions and @throws are synonymous Javadoc tags used to define the exceptions that may be thrown by an operation.

To use these tags you can proceed as follows:

- From the operation property sheet, click the Implementation tab and click the Exceptions tab to open the Exceptions page. You can type exceptions in this page.

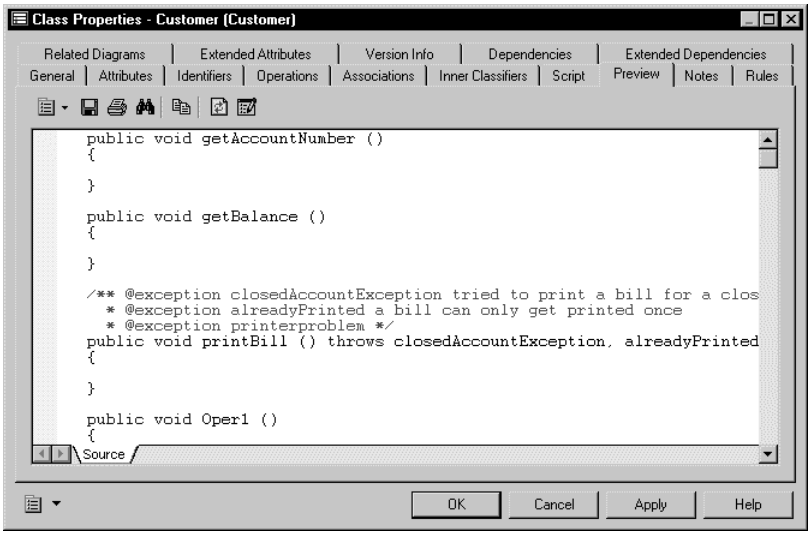


- In the same property sheet, click the Extended Attributes tab, select the Javadoc@exception line in the list and click the ellipsis button in the Value column. Type values for each declared exception, do not forget to repeat @exception or @throws before each exception.


Text
@exception closedAccountException tried to print a bill for a closed (he
@exception alreadyPrinted a bill can only get printed once
OK
Cancel

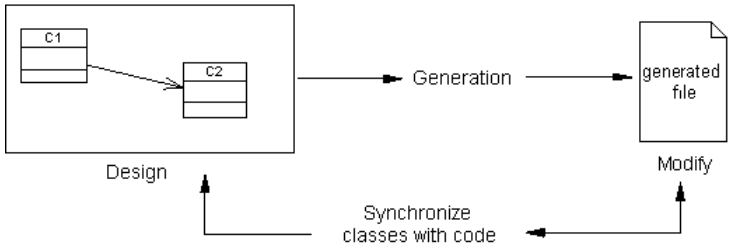
It is also possible to type values directly after the @exception or the @throws tags in the Extended Attributes page. These comments describe exceptions that are not listed in the operation exceptions, and do not appear after the **`throws`** parameter in the Preview page of the class.

When you preview the generated code, each exception appears with its value:



Synchronizing a model with generated files

You can use an accelerated process in class diagrams to modify or add classes by accessing generated files. If you generate code, you can use your code editor to visualize and change the generated file. This feature allows you to design in PowerDesigner, use the generation process, then visualize and modify the generated file in the code editor, synchronize the classifiers with the Java source code and then go back to the model. With this feature, you can modify the generated file and reverse in the same generated file.



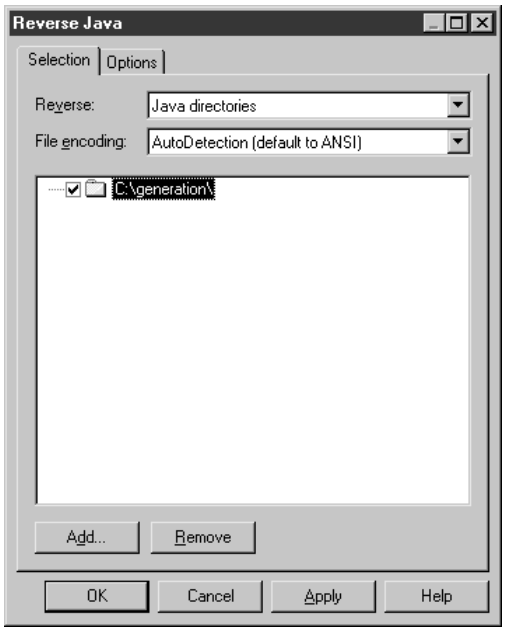
The synchronization launches a reverse engineering dialog box, pre-selects the Java Directories option, and fills the list of classifiers with the classifiers selected in the class diagram.

You can then easily locate the files that should be taken into account for synchronization. If there is no classifier selected, the reverse feature pre-selects the Java Directories option and adds the current directory to the list.

❖ **To synchronize a model with generated files:**

1 Select Language→Synchronize with generated files to display the Reverse Java dialog box.

The Selection page appears.



2 Select another Java format from the Reverse dropdown listbox if needed.

3 Select another file encoding type from the File Encoding dropdown listbox if needed.

4 Click the Add button.

The Browse for Folder dialog box appears.

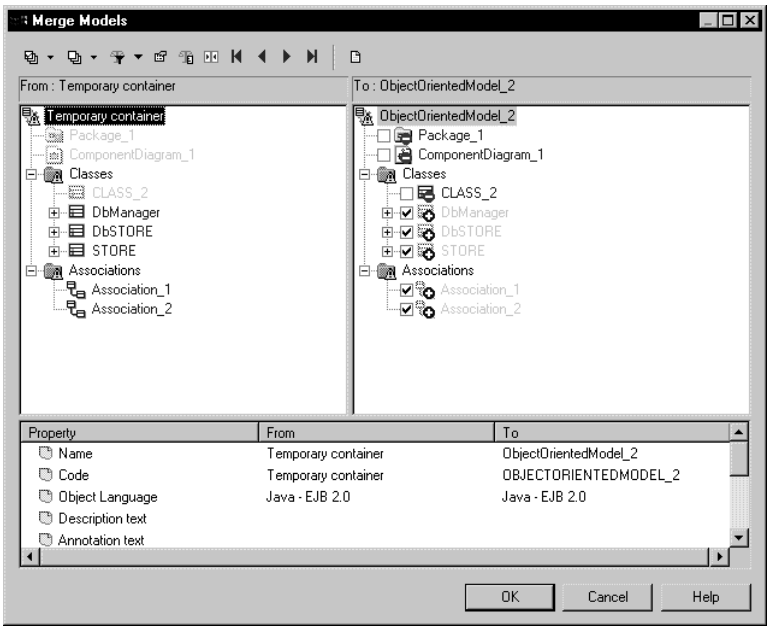
5 Select the directory you need.

6 Click OK.

The Reverse Java dialog box displays your selection.

7 Click OK.

A progress box appears, followed by the Merge Models dialog box.



Merging models
The Merge Models dialog box shows the From Model (source directory) in the left pane, and the To Model (current model) in the right pane. You can expand the nodes in the To Model pane to verify that the merge actions selected correspond to what you want to perform.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

8 Click OK.

The Reverse page of the Output window displays the changes which occurred during synchronization and the diagram window displays the synchronized model.

Reverse engineering PowerBuilder

You can reverse engineer PowerBuilder NVO (non-visual objects) into an OOM from the following sources:

- PowerBuilder application
- SRU file (user script from PowerBuilder)

For each reversed PowerBuilder object, a class is created in the model with the same name and containing the same information. When you reverse engineer an object with the same name as a class that already exists in a model, you can choose in the displayed Merge Model window, either to replace the existing class, or to keep the existing class definition in the model.

You can reverse only the following PowerBuilder User Objects:

- Custom Class
- Standard Class
- Custom Visual
- External Visual
- Standard Visual

Reverse engineering PowerBuilder options

You define the following PowerBuilder reverse engineering options from the Reverse PowerBuilder dialog box:

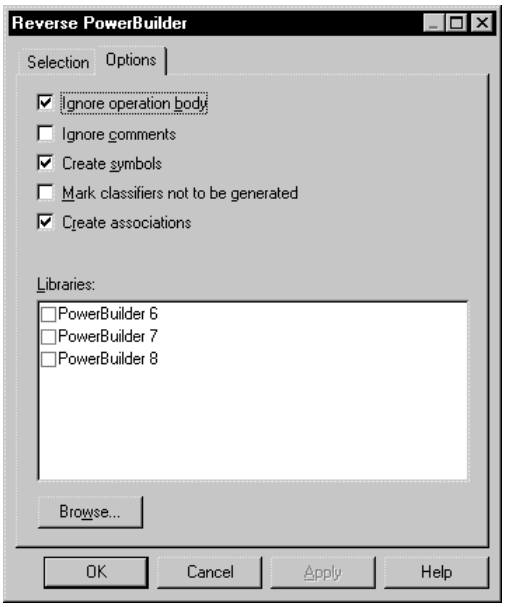
Option	Result of selection
Ignore operation body	Reverses PowerBuilder objects without including the body of the code
Ignore comments	Reverses PowerBuilder objects without including code comments
Create symbols	Creates a symbol for each object in the diagram. Otherwise, reversed objects are visible only in the browser
Mark classifiers not to be generated	Reversed classifiers (classes and interfaces) will not be generated from the model. To generate the classifier, you must select the Generate check box in its property sheet
Create Associations	Creates associations between classes and/or interfaces

Option	Result of selection
Libraries	PowerBuilder models to be loaded in the workspace. The Setup program installs these models that contain the class libraries of each version of PowerBuilder. You can load them quickly into PowerDesigner thus save time reversing them

❖ **To define PowerBuilder reverse engineering options:**

1 Select Language→Reverse Engineer PowerBuilder to display the Reverse PowerBuilder dialog box.

2 Click the Options tab to display the Options page.



3 Select or clear options.

4 Click Apply and Cancel.

Loading a PowerBuilder library model in the workspace

When you reverse engineer PowerBuilder files, you can, at the same time, load one of the PowerBuilder models that contains the class libraries of a particular version of PowerBuilder. The Setup program installs these models in the Library directory.

You can select to reverse a PowerBuilder library model from the Options page of the Reverse PowerBuilder dialog box.

You can open a PowerBuilder library model in the workspace from the Library directory.

❖ **To load a PowerBuilder library model:**

1 Select File→Open to display the Open dialog box.

2 Select or browse to the Library directory.

The available library files are listed. Each PB file corresponds to a particular version of PowerBuilder.

3 Select the file PB8.OOM

This file contains all the library class files of PowerBuilder version 8.

4 Click Open.

The OOM opens in the workspace.

Reverse engineering objects from a PowerBuilder application

When you reverse engineer objects from a PowerBuilder application, you can select only one PowerBuilder application from the PB application dropdown listbox. You can then add objects that belong to this application to the list of objects to reverse.

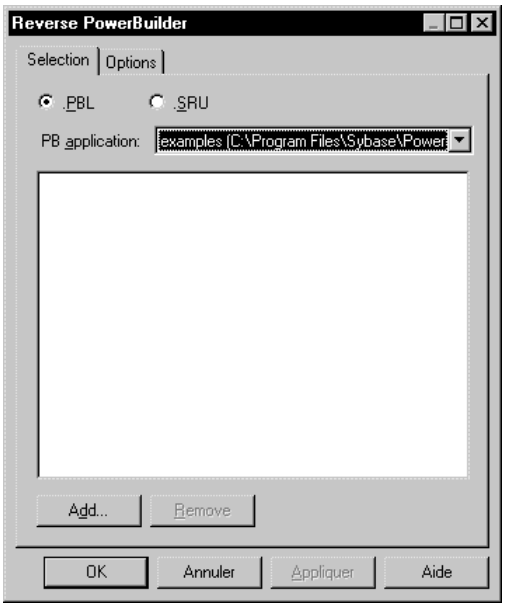
If PowerBuilder is not installed on your machine, you cannot generate objects for a PowerBuilder application, and you can only reverse .SRU files.

Reverse engineered PowerBuilder objects always keep their original names.

❖ **To reverse PowerBuilder objects from a PowerBuilder application:**

1 Select Language→Reverse Engineer PowerBuilder to display the Reverse PowerBuilder dialog box.

2 Select the .PBL radio button in the Selection page.



3 Select a PowerBuilder application from the PB Application dropdown listbox.

4 Click the Add button.

A standard Open dialog box appears.

5 Select the .PBL files you want to reverse and click Open.

> **Multi-selection**
> You select several files simultaneously using the CTRL or SHIFT keys.

The Reverse PowerBuilder dialog box displays the files you selected.

6 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

The classes are added to your model. They are visible in the diagram and in the Browser. They are also listed in the Reverse page of the Output window, located in the lower part of the main window.

Reverse engineering objects from SRU files

SRU files are text files containing the definition of PowerBuilder User Objects.

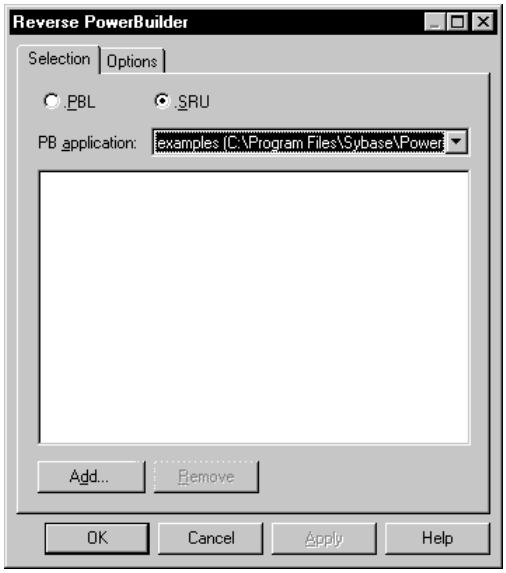
You do not need PowerBuilder installed on your machine to reverse engineer objects contained in SRU files.

Reverse engineered PowerBuilder objects always keep their original names.

❖ **To reverse engineer PowerBuilder objects from SRU files:**

1 Select Language→Reverse Engineer PowerBuilder to display the Reverse PowerBuilder dialog box.

2 Select the .SRU radio button in the Selection page.



3 Click the Add button.

A standard Open dialog box appears.

4 Select the .SRU files that you want to reverse and click Open.

> **Multi-selection**
> You select several files simultaneously using the CTRL or SHIFT keys.

The Reverse PowerBuilder dialog box displays the files you selected.

5 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

The classes are added to your model. They are visible in the diagram and in the Browser. They are also listed in the Reverse page of the Output window, situated in the lower part of the main window.

Reverse engineering XML

You can reverse engineer the following types of XML files into an OOM:

- XML-DTD Provides an overall structure for an XML file in DTD format
- XML-Schema Provides an overall structure for an XML file in Schema format
- XML-Data Is used with BizTalk

XML-DTD

When you reverse engineer a DTD file:

- Referenced DTD files, using DOCTYPE keyword, may not be reversed correctly, it is therefore better to choose a real DTD file rather than an XML document referencing a DTD file with a DOCTYPE document type declaration
- Elements of type #PCDATA are reversed as attributes
- An element that has both a parent and a child element is linked to its parent element by an aggregation link
- If an empty element has no child object but has attributes, it is reversed as a class and its attributes become attributes of the class
- Attributes of type ID and IDREF(S) are reversed as attributes with ID and IDREF(S) data types
- The attribute sequence order may not be preserved
- Attribute groups structure is not preserved
- DTD files may not be reversed properly if they contain some reference to an undefined parameter entity

When you reverse a DTD file into an OOM, you get a more readable view of the DTD. This feature can be very helpful when you want to check and understand a new DTD that you have not generated.

XML-Schema

When you reverse engineer an XML-Schema file, the following main elements are not reversed:

- Namespace
- Key/keyref
- Field/selector
- Unique
- Union

Moreover, the sequence order of the contents of an element may not be preserved after reverse engineering.

The following table maps XML-Schema objects to OOM objects when reverse engineering an XML-Schema file. It also lists the stereotypes of these objects to better identify them:

XML-Schema object	OOM object	Stereotype
Schema	Model, package	—
Global element	Class	<<element>>
Global attribute	Class	<<attributeGroup>>
Global simple type	Class	<<simpleType>>
Global complex type	Class	<<complexType>>
Simple type	Attribute	<<simpleType>>
Nested complex type	Class	<<complexType>>

XML-Data

When you reverse engineer an XML-Data file, the following elements are processed:

- If an <ElementType> contains sub-elements, it is reversed as a class, otherwise, it is reversed as an attribute
- An <element> is reversed as an association if its type is a class, otherwise, it is reversed as an attribute
- <AttributeType> and <attribute> are reversed as attributes with the <<attribute>> stereotype

Reverse engineering XML options

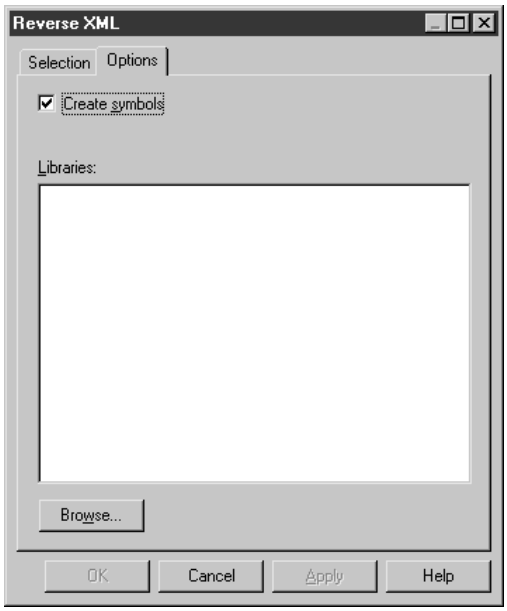
You define the following XML reverse engineering option from the Reverse XML dialog box:

Option	Result of selection
Create symbols	Creates a symbol for each reversed XML object in the diagram. Otherwise, reversed objects are visible only in the Browser
Libraries	Allows you to load libraries while reversing. The default list of libraries is located in the Library directory of your installation

❖ **To define XML reverse engineering options:**

1 Select Language→Reverse Engineer XML to display the Reverse XML dialog box.

2 Click the Options tab to display the Options page.



3 Select or clear the Create Symbols check box.

4 Browse to the Library directory, if required.

5 Click Apply and Cancel.

Reverse engineering XML files

You use the following procedure if you want to reverse XML files into an OOM:

❖ **To reverse engineer XML files:**

1 Select Language→Reverse Engineer XML to display the Reverse XML dialog box.

2 Click the Add button in the Selection page.

A standard Open dialog box appears.

3 According to your object language, select the .xsd, .xdr or .dtd files you want to reverse and click Open.

> **Multi-selection**
> You select several files simultaneously using the CTRL or SHIFT keys.

The Reverse XML dialog box displays the files you selected.

4 Click OK.

A progress box appears. If the model in which you are reverse engineering already contains data, the Merge Models dialog box appears.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

The classes are added to your model. They are visible in the diagram and in the Browser. They are also listed in the Reverse page of the Output window, located in the lower part of the main window.

Reverse engineering into a new OOM

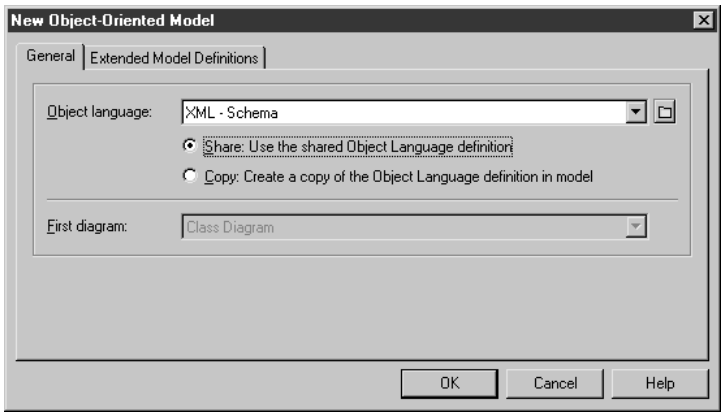
You can reverse engineer object language files (Java, PowerBuilder, or XML) into a new OOM.

❖ **To reverse engineer object language files into a new OOM:**

1 Select File→Reverse Engineer→Object Language to display the New Object-Oriented Model dialog box.

2 Click the Share radio button.

3 Select an object language in the dropdown listbox.



4 <optional> When you reverse into a new OOM, you can select one or several extended model definitions to reverse at the same time by clicking the Extended Model Definitions tab, and select the extended model definitions of your choice.

For more information on extended model definitions, see chapter Extended Model Definitions Reference Guide, in the *Advanced User Documentation.*

5 Click OK.

Depending on the object language the corresponding dialog box appears to let you select files and reverse options.

6 Click OK to start reverse engineering.

A progress box appears. The classes are added to your model. They are visible in the diagram and in the Browser. They are also listed in the Reverse page of the Output window, located in the lower part of the main window.

This product includes XML4C 3.0.1 software developed by the Apache Software Foundation (http://www.apache.org/)
Copyright (c) 1999 The Apache Software Foundation. All rights reserved.
THE XML4C 3.0.1 SOFTWARE ("SOFTWARE") IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER 15

Generating for a Language

About this chapter

This chapter describes how to generate objects from an OOM.

Contents

Topic	Page
Generation options for languages	620
Generating for C++	626
Generating for C#	630
Generating for Java	633
Generating for PowerBuilder	640
Generating for VB.NET	643
Generating for XML	646

Generation options for languages

You can generate the following types of objects for the following supported languages in an OOM:

Object language	What is generated
Analysis	No files generated as this language is mainly used for modeling purpose
C#	.CS definition files
C++	C++ definition files(.h and .cpp)
Java	Java files from classes and interfaces of the model. Includes support of EJB and J2EE
PowerBuilder	.PBL application or .SRU files from classes of the model
Visual Basic.Net	.VB files
XML–Data (BizTalk)	.XDR files
XML–DTD	.DTD files
XML–Schema	.XSD files. Includes standard XML language properties

User-defined generation

You can use the Generation Template Language (GTL) mechanism to produce user-defined text for code generation. It allows you to access all elements of the model and to customize the generation process. It is available for all object languages supported in an OOM, except PowerBuilder.

For more information on the Generation Template Language mechanism, see section Defining Generation Template Language, in chapter Generation Reference Guide in the *Advanced User Documentation*.

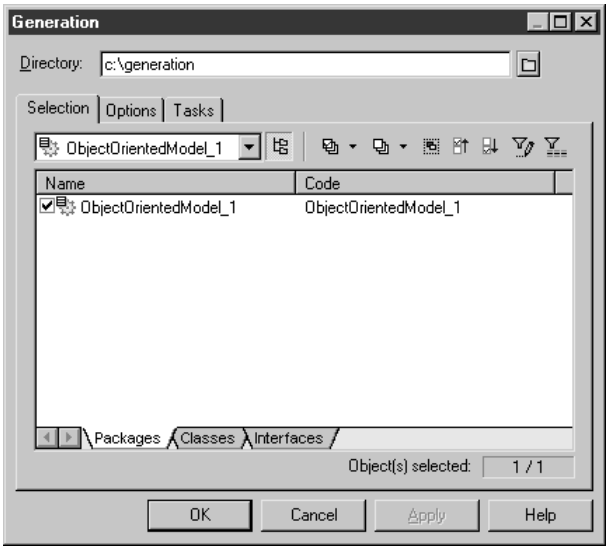
Trace mode in Preview pages

An Enable Trace Mode check box allows you to display the generation templates used to generate code from the Preview page in object property sheets.

For more information on the trace mode property, see section Object language properties, in chapter Object Languages Reference Guide in the *Advanced User Documentation*.

Selecting objects to generate

You select objects for generation from the Selection page of the Generation dialog box. The selection page contains tabbed pages for selecting specific objects of the model.



Listing objects contained in a model or package

You can display objects in the current model, or objects in individual packages contained in the model. If you select the Include Sub-Packages tool, you can display either all objects in the current model, or all objects in a package.

The selection options are the following:

Parent object	Include Sub-packages tool selected	Include Sub-packages tool not selected
Model	All objects in the model including all objects contained in packages and sub-packages	All objects in the model except objects contained in packages and sub-packages
Package	All objects contained in the package including all objects contained in sub-packages	All objects in the package except objects contained in sub-packages

Defining generation options

You can check your model before generation by using the Check Model check box. This check box is available for all languages. The generation stops if an error is found: you must correct this error before starting the generation again.

For more information on the Check Model feature, see section Checking an OOM in chapter Working with Object-Oriented Models.

If you do not see any option in the Options page of the Generation dialog box, it means that no option has been defined in the corresponding object language (.XOL file).

You can however customize your object language at any time, and add any options you may need during generation. For example, you may want to sort the order in which the operations are displayed according to their visibility. You define generation options in the Generation/Options category of the object language available from Language→Edit Current Object Language.

For more information on how to add options for generation, see section Options category, in chapter Object Languages Reference Guide in the *Advanced User Documentation*.

Defining generation tasks

The Tasks page contains standard commands defined in the object language. You can select any tasks to execute at generation time.

If you do not see any task in the Tasks page of the Generation dialog box, it means that no command has been defined in the corresponding object language (.XOL file).

You can however customize tasks for your current language so that, during generation, your files are automatically compiled, or packaged in an archive for example. You define generation tasks and commands in the Generation/Tasks category and the Generation/Commands category of the object language available from Language→Edit Current Object Language.

For more information on how to add tasks, see section Commands category, and section Tasks category, in chapter Object Languages Reference Guide in the *Advanced User Documentation*.

Using the Move up or down arrows
You can easily modify the order of commands execution using the Move up or Move down arrows. You can only select one value check box at a time.

Defining generation targets (XEM)

When you start the generation process, you can choose to:

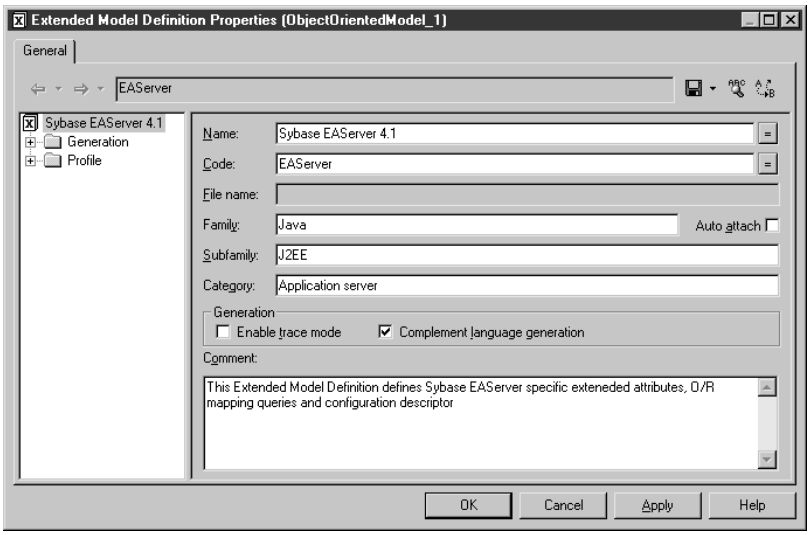
- Generate for a selected object language
- Use an extended model definition that extends the object language

To use an extended model definition that extends the object language, you must have an extended model definition attached to the model. You can either create a new extended model definition or import an existing extended model definition in the model.

For more information on how to create or import an extended model definition, see sections Creating an extended model definition, or Importing an extended model definition into a model, in chapter Extended Model Definitions Reference Guide in the *Advanced User Documentation*.

If you use an extended model definition to extend the object language, you must use an extended model definition designed for generation.

To do this, you must verify that the value Complement Language Generation is selected in the extended model definition editor.



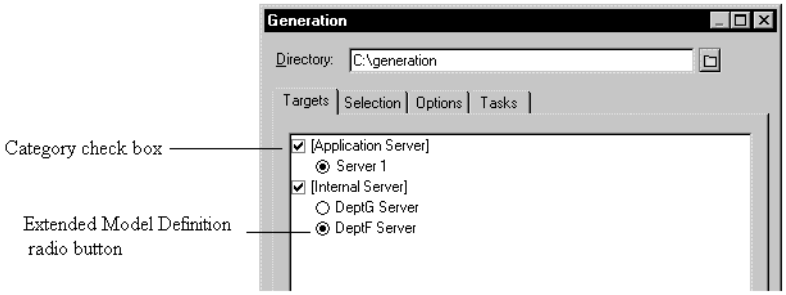
When you attach an extended model definition to your model, the Targets page in the Generation dialog box displays the target extended model definition that can be selected for generation.

For more information on the generation targets, see section Generation targets, in chapter Extended Model Definitions Reference Guide, in the *Advanced User Documentation*.

The Targets page contains category check boxes and radio buttons.

The check boxes allow you to select a server category that may contain multiple extended model definitions, whereas the radio buttons allow you to select extended model definitions.

The radio buttons are mutually exclusive, as it is only possible to select one extended model definition at a time.



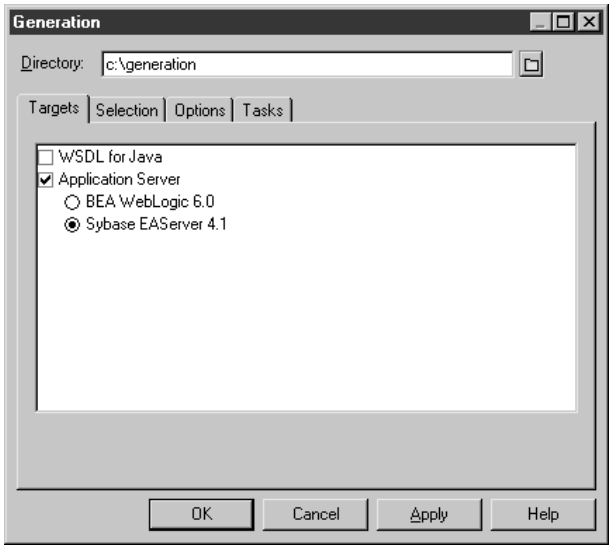
❖ **To define a generation target:**

1 Select Language→Generate *Object Language*.

The Generation dialog box opens to the Targets page.

2 Select or browse to a directory that will contain the generated files.

3 Select the extended model definition that you want to use for generation.



4 Click Apply if you want to generate at a later time.
or
Click OK to generate.

Generating for C++

This section explains some of the features of the C++ object language in order to generate C++ code from objects in the OOM.

C++ implementation details

The extended attribute 'UseNamespace' has been added to classes in order to specify whether the class should be declared inside the package namespace.

Unsupported ANSI C++ features

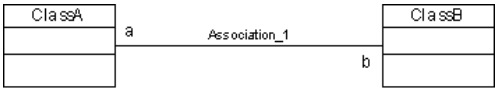
PowerDesigner does not support the following C++ features:

- Templates
- Enums
- Typedefs
- Inline methods

Managing bidirectional associations

The problem of bidirectional associations is addressed by using forward declarations instead of includes.

Consider a bidirection association between ClassA and ClassB.



The generated code in A.h would be the following:

```
#if !defined(__A_h)
#define __A_h


class B;    // forward declaration of class B


class A
{
public:
  B* b;
```

```
protected:
private:

};

#endif
```

The generated code in B.h would be the following:

```
#if !defined(__B_h)
#define __B_h

class A;    // forward declaration of class A

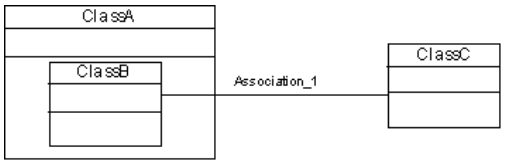
class B
{
public:
  A* a;

protected:
private:

};

#endif
```

This approach will not work if one of the classes is an inner class because it is not possible to forward-declare inner classes in C++.



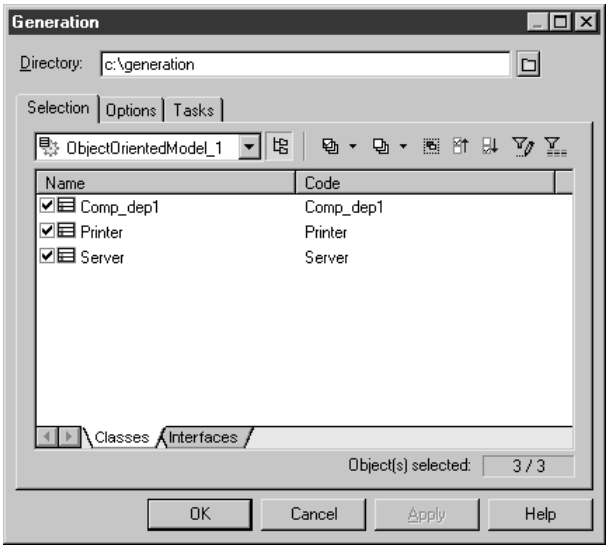
If such a situation occurs, a warning message appears during generation, and the corresponding code is commented out.

Generating C++ files

When generating with C++, the files generated are generated for classes and interfaces. A header file with the .h extension, and a source file with the .cpp extension are generated per classifier. A generation log file is also created after generation.

❖ **To generate C++ files:**

1 Select Language→Generate C++ Code to display the Generation dialog box.



2 Type a destination directory for the generated file in the Directory box.
or
Click the Select a Path button to the right of the Directory box and browse to select a directory path.

3 Select the objects to include in the generation from the tabbed pages at the bottom of the Selection page.

Select Tools
All classes of the model, including those grouped into packages, are selected and displayed by default. You use the Select tools to the right of the Folder Selection dropdown listbox to modify the selection. The Include Sub-Packages tool allows you to include all classes located within packages.

4 Click the Options tab to display the Options page.

5 <optional> Select the Check Model check box if you want to verify the validity of your model before generation.

6 Select a value for each required option.

7 Click the Tasks tab to display the Tasks page.

8 Select the required task(s).

9 Click OK to generate.

A Progress box appears. The Result list displays the files that you can edit. The result is also displayed in the Generation page of the Output window, located in the bottom part of the main window.

All C++ files are generated in the destination directory.

Generating for C#

This section explains some of the features of the C# object language in order to generate C# code from objects in the OOM.

Selecting C# generation options

You can set the following option, available from the Options page of the Generation dialog box in C#:

Options	Description
Generate Web Service C# code in .ASMX file	Generates the C# code in the .ASMX file

Selecting C# generation tasks

The Tasks category contains the following commands:

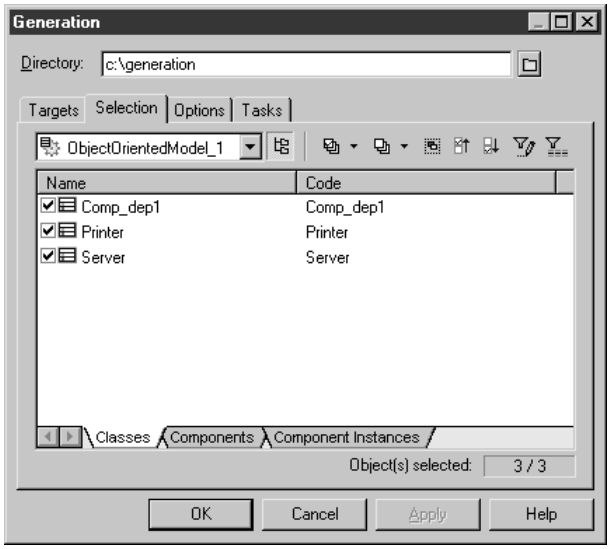
Options	Description
Compile source files	Compiles the source files
Generate Web service proxy code (WSDL)	Generates the proxy class

Generating C# files

The files generated are generated with the .cs extension. A generation log file is also created after generation.

❖ **To generate C# files:**

1 Select Language→Generate C# Code to display the Generation dialog box.



2 Type a destination directory for the generated file in the Directory box.
or
Click the Select a Path button to the right of the Directory box and browse to select a directory path.

3 Click the Selection tab to display the Selection page.

4 Select the objects to include in the generation from the tabbed pages at the bottom of the page.

> **Select Tools**
> All classes of the model, including those grouped into packages, are selected and displayed by default. You use the Select tools to the right of the Folder Selection dropdown listbox to modify the selection. The Include Sub-Packages tool allows you to include all classes located within packages.

5 Click the Options tab to display the Options page.

6 <optional> Select the Check Model check box if you want to verify the validity of your model before generation.

7 Select a value for each required option.

8 Click the Tasks tab to display the Tasks page.

9 Select the required task(s).

10 Click OK to generate.

A Progress box appears. The Result list displays the files that you can edit. The result is also displayed in the Generation page of the Output window, located in the bottom part of the main window.

All C# files are generated in the destination directory.

Generating for Java

This section explains some of the features of the Java object language in order to generate Java code from objects in the OOM.

EJB subfamily

If you work on a Java OOM created in version 9.0, the subfamily of Java is EJB 2.0 and it supports EJB. In version 9.5, the subfamily of Java has become J2EE, it supports J2EE components (servlets, JSPs **and** EJB).

You should use the new Java object language provided with the current version to use J2EE components. If you had customized the Java object language in the previous version, you have to apply these changes manually in the new Java object language.

What tools do you need?

You need to set the Java executables you need (for example jar, java, javac, javadoc, J2EE verifier) in the Variables page of the General Options feature (Tools menu) in order to use them during the generation process.

Generating comments using Javadoc

The generation process generates object comments compliant with Javadoc.

Javadoc is a tool delivered in the JDK that parses the declarations and documentation comments in a set of Java source files and produces a corresponding set of HTML pages describing model objects.

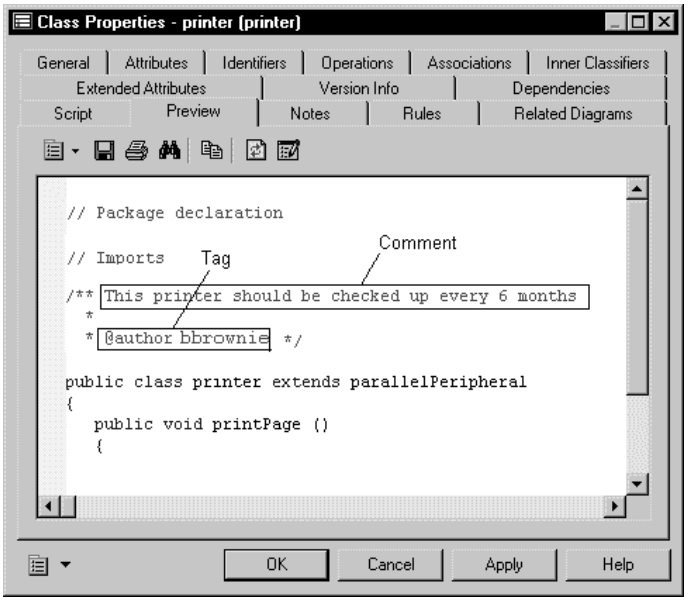
Javadoc comments

Javadoc comments are included in the source code of an object, immediately before the declaration of any object, between the characters /** and */.

A Javadoc comment can contain:

- A description after the starting delimiter /**. This description corresponds to a Comment in OOM objects
- Tags prefixed by the @ character

For example, in the following code preview page, you can read the tag @author, and the comment inserted from the Comment box in the General page of the class property sheet.



The following table explains the meaning of the Javadoc comments:

Javadoc comment	Description	Applies to
@since	Adds a “Since” heading with the specified since-text to the generated documentation	Class, interface, operation, attribute
@deprecated	Adds a comment indicating that this API should no longer be used	Class, interface, operation, attribute
@author	Adds an Author entry	Class, interface
@version	Adds a Version entry, usually referring to the version of the software	Class, interface
@see	Adds a “See Also” heading with a link or text entry that points to reference	Class, interface, operation, attribute
@return	Adds a “Returns” section with the description text	Operation

Javadoc comment	Description	Applies to
@throws	Adds a “Throws” subheading to the generated documentation	Operation
@exception	Adds an “Exception” subheading to the generated documentation	Operation
@serialData	Documents the types and order of data in the serialized form	Operation
@serialField	Documents an ObjectStreamField component of a Serializable class’ serialPersistentFields member	Attribute
@serial	Used in the doc comment for a default serializable field	Attribute
@param	Adds a parameter to the Parameters section	Operation

Javadoc comments that are reversed in the Java code of your OOM objects are also preserved.

For more information on the reverse engineering of Javadoc comments, see section Reverse engineering comments using Javadoc in chapter Reverse engineering.

Selecting Java generation options

You can set Java generation to sort the order in which the attributes and operations of the class in a Java class definition file are displayed. You can sort the order according to either visibility or member type for example.

You can also set the following options, available from the Options page of the Generation dialog box in Java:

Options	Description
Sort class members primarily by	Sorts attributes and operations by type or visibility
Generate package imports	Used to declare import of the whole package
Class members type sort	Sorts attributes and operations type
Class members visibility sort	Sorts attributes and operations visibility
Generate CMP field accessors in component interfaces (EJB)	Generates CMP fields getter and setter operations to EJB interfaces

Options	Description
Generate CMR field accessors in remote interfaces (EJB)	Generates CMR fields getter and setter declarations in remote interfaces
Add Java classes source code in the JAR file (EJB)	Includes Java classes code in the JAR
Jar Web component classes (J2EE)	Archives Web component classes in a Jar
XRPCC input (WSDL)	Tool that generates WSDL and proxy for the Web service Java platform

You can set Java generation options to sort the attributes and operations of the class in a Java class definition file.

Primary sort

You can sort the order according to the visibility or the type.

Option	Result of selection
Visibility	Public attributes and operations are generated before private attributes and operations in the Java file
Type	Attributes and operations are sorted by type whatever their visibility

Visibility sort

You can sort the visibility order as follows:

Option	Result of selection
Public – Private	Public attributes and operations are generated before private attributes and operations
Private – Public	Private attributes and operations are generated before public attributes and operations
None	Attributes and operations order remains unchanged

Type sort

You can sort the type order as follows:

Option	Result of selection
Attributes – Operations	Class attributes are generated before the operations
Operations – Attributes	Class operations are generated before the attributes

Generate package imports

When a class is used by another class, it is referenced by a class import:

```
import package1.package2.class.
```

The Generate package imports allows you to declare import of the whole package, it saves time whenever many classes of the same package are possibly referenced:

```
import package1.package2.*;
```

Selecting Java generation tasks

The following commands are available from the Tasks page in the Generation dialog box for Java:

Command	Description
Compile Java sources	Starts a compiler using the javac command to compile Java source files
Compile Java sources, run Java application	Starts a compiler and run the Java application using the java command
Generate Javadoc	Generates Javadoc
Build EJB, create EJB Jar	Compiles component sources, creates a jar file containing the EJB and deployment descriptor
Build EJB, create EJB Jar, Run J2EE verifier	Compiles component sources, creates a jar file containing the EJB and deployment descriptor and checks the generated jar validity
Build Web component, create Web application WAR	Compile component sources, creates a WAR file containing the Web service component and deployment descriptor
Create EJB JARs, WARs and EAR	Creates a JAR with EJB and deployment descriptor, creates a WAR with servlet or JSP and deployment descriptor and an EAR file containing the WAR and JAR files generated for the model, and an application descriptor
Generate Web service proxy code using XRPCC tool - client side (WSDL)	Generates the proxy class using the XRPCC tool
Generate WSDL using XRPCC tool - server side (WSDL)	Generates the WSDL file using the XRPCC tool

Note: the default compiler command is the `javac` executable. The default run command for the Java compiler is the `java` executable.

Those commands are used by default in PowerDesigner but you can customize your commands using any other suitable executable by selecting Tools→General Options→Variables and setting your preferences in this page.

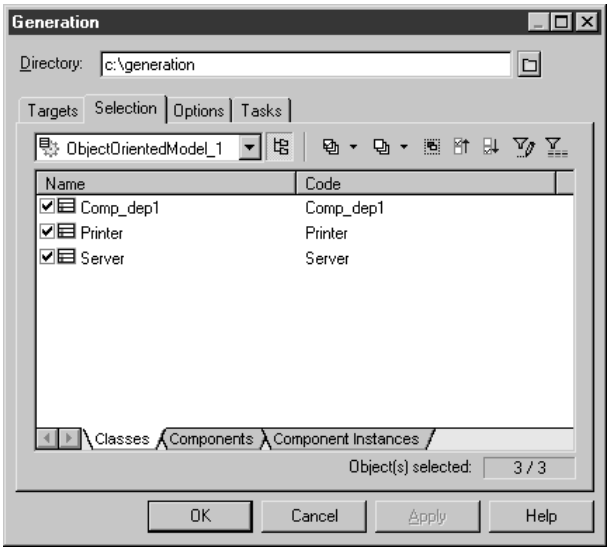
Generating Java files

You generate Java source files from the classes and interfaces of a model. A separate file, with the file extension .java, is generated for each class or interface that you select from the model. A generation log file is also created at the same time. You can only generate Java files from one model at a time.

A Java source can contain several classes, whereas a Java .class can contain only one class. You can compile the .java class files that you generate from an OOM in any Java compiler tool and run your Java application.

❖ **To generate Java files:**

1 Select Language→Generate Java Code to display the Java generation dialog box.



2 Type a destination directory for the generated file in the Directory box.
or
Click the Select a Path button to the right of the Directory box and browse to select a directory path.

3 Click the Selection tab to display the Selection page.

4 Select the objects to include in the generation from the tabbed pages at the bottom of the page.

> **Select Tools**
> All classes of the model, including those grouped into packages, are selected and displayed by default. You use the Select tools to the right of the Folder Selection dropdown listbox to modify the selection. The Include Sub-Packages tool allows you to include all classes located within packages.

5 Click the Options tab to display the Options page.

6 <optional> Select the Check Model check box if you want to verify the validity of your model before generation.

7 Select a value for each required option.

8 Click the Tasks tab to display the Tasks page.

9 Select the required task(s).

10 Click OK to generate.

A Progress box appears. The Result list displays the files that you can edit. The result is also displayed in the Generation page of the Output window, located in the bottom part of the main window.

All Java files are generated in the destination directory.

Generating for PowerBuilder

This section explains some of the features of the PowerBuilder object language in order to generate specific code from objects in the OOM.

Selecting PowerBuilder generation options

PowerBuilder is a specific language that uses its own generation options. You can generate PowerBuilder NVOs (non-visual objects) from the classes of an OOM to either of the following:

- A PowerBuilder application (PBL)
- SRU files

You can create PowerBuilder user objects only from classes, not from interfaces.

You can set the following PowerBuilder generation options in the Options page of the PowerBuilder User Object Generation dialog box from Language→Generate PowerBuilder:

Option	Result of selection
Check model	Checks the model before generation and stops generation if an error is found
PBL	PowerBuilder library and application into which PowerDesigner generates
SRU	Directory into which PowerBuilder NVOs .SRU files are generated

PBL application

You can generate PowerBuilder NVOs from an OOM and use them in PowerBuilder. If PowerBuilder is not installed, you cannot generate objects for a PowerBuilder application.

When generating objects for a PowerBuilder application. You must specify both PowerBuilder library and application that use the objects in the Options page, otherwise you are not able to use them in PowerBuilder.

Option	Result of selection
PB library	Library into which PowerDesigner generates PowerBuilder objects (NVOs)

Option	Result of selection
PB application	PowerBuilder application into which you generate PowerBuilder NVOs. If PowerBuilder is not installed, no application is listed

SRU files

You can generate NVOs from the classes in an OOM. When you generate PowerBuilder objects in SRU files, a separate file with the extension .SRU is created for each selected class in the OOM. Each file contains a NVO corresponding to the definition of each class in the OOM. You do not need to install PowerBuilder to generate SRU files.

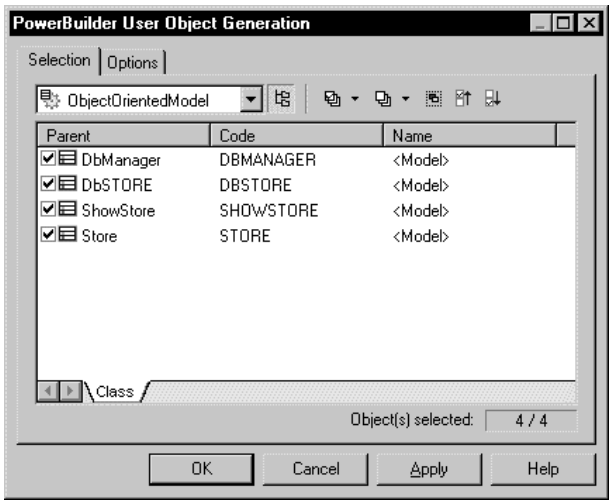
Selecting PowerBuilder objects to generate

When generating objects for a PowerBuilder application, you must fill in both PowerBuilder library and application fields in the Options page.

For more information on PowerBuilder options, see section Selecting PowerBuilder generation options.

❖ **To generate objects for PowerBuilder:**

1 Select Language→Generate PowerBuilder to display the PowerBuilder User Object Generation dialog box.



2 Select a model or package from the Folder Selection dropdown listbox.

3 Select the classes to generate from the list.

Select Tools
All classes of the model are selected and displayed by default. Use the Select tools to the right of the Folder Selection dropdown listbox to modify the selection. The Include Sub-Packages tool allows you to include all objects located within packages.

4 Click the Options tab to display the Options page.

5 <optional> Select the Check Model check box if you want to verify the validity of your model before generation.

6 Select the required options.

7 Click OK to generate.

A PowerBuilder user object is generated in the PowerBuilder application for each selected class.

Generating for VB.NET

This section explains some of the features of the VB.NET object language in order to generate VB code from objects in the OOM.

Selecting VB.NET generation options

You can set the following option, available from the Options page of the Generation dialog box in VB.NET:

Options	Description
Generate VB.NET Web service code in .ASMX file instead of .VB file	Generates the Visual Basic code in the .ASMX file

Selecting VB.NET generation tasks

The Tasks category contains the following commands:

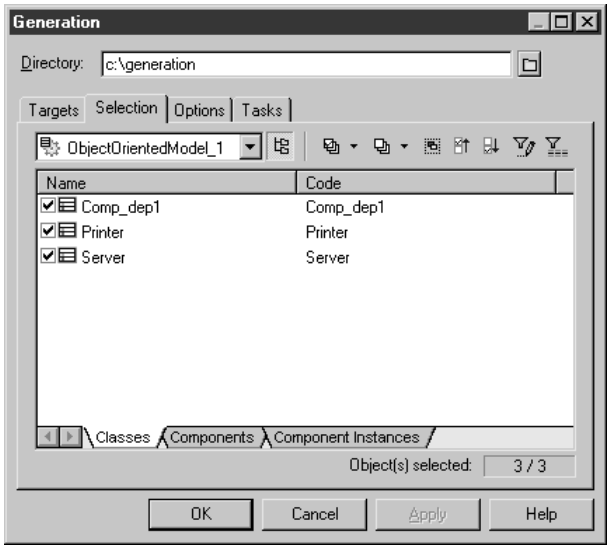
Options	Description
Generate Web service proxy code (WSDL)	Generates the proxy class
Compile source files	Compiles the source files

Generating VB.NET files

The files generated are generated with the .vb extension. A generation log file is also created after generation.

❖ **To generate VB.NET files:**

1 Select Language→Generate VB.NET Code to display the Generation dialog box.



2 Type a destination directory for the generated file in the Directory box.
or
Click the Select a Path button to the right of the Directory box and browse to select a directory path.

3 Click the Selection tab to display the Selection page.

4 Select the objects to include in the generation from the tabbed pages at the bottom of the page.

> **Select Tools**
> All classes of the model, including those grouped into packages, are selected and displayed by default. You use the Select tools to the right of the Folder Selection dropdown listbox to modify the selection. The Include Sub-Packages tool allows you to include all classes located within packages.

5 Click the Options tab to display the Options page.

6 <optional> Select the Check Model check box if you want to verify the validity of your model before generation.

7 Select a value for each required option.

8 Click the Tasks tab to display the Tasks page.

9 Select the required task(s).

10 Click OK to generate.

A Progress box appears. The Result list displays the files that you can edit. The result is also displayed in the Generation page of the Output window, located in the bottom part of the main window.

All VB files are generated in the destination directory.

Generating for XML

This section summarizes generation options used when generating for the following XML languages:

- XML-Schema
- XML-DTD
- XML-Data

To change the XML format type, you must change the object language for the model in Language→Change Current Object Language.

You can create a new XML object language based on an existing one if you want to generate in another type of XML format that is different to those available in PowerDesigner.

Generating for XML-Schema

You can generate an XML file in XML-Schema: a .XSD file will be generated per package or model. You can generate XML-Schema definition in this specific format for validations and/or data exchange purposes.

Navigable associations are migrated and generated as attributes, although they do have their own definition in the object language file.

Interfaces, operations, and links (dependencies, realizations and generalizations) are not included in the generated file.

Using stereotypes in XML-Schema

To generate an object with a specific syntax, you have to set its stereotype to a selected value. It can be one of your own: myStereotype for instance; you define a "<<myStereotype>> Definition" entry in the corresponding object sub-category under the Settings\Stereotypes\Objects category of the Object Language, from Language→Edit Current Object Language.

If a class has no stereotype, or if a stereotype has no specified definition, the default definition is used.

Generating for XML-DTD

You can generate an XML file in XML-DTD: a .DTD file will be generated per package or model. A DTD file provides an overall structure for an XML file. You can generate XML-DTD definition in this specific format for validations and/or data exchange purposes.

Navigable associations are migrated and generated as attributes, although they do have their own definition in the object language file.

Interfaces, operations, and links (dependencies, realizations and generalizations) are not included in the generated file.

Generating for XML-Data

You can generate an XML file in XML-Data: a .XDR file will be generated per package or model.

Navigable associations are migrated and generated as attributes, although they do have their own definition in the object language. Interfaces, operations, and links (dependencies, realizations and generalizations) are not included in the generated file.

Selecting XML generation options

You can set the following options, available from the Options page of the Generation dialog box in XML-Data:

Option	Default value
GenerateCollections	Generates a class collection for each class of the model (XML-DATA only)

For example, if you create a Customer class with a code named **CUSTOMER**, the generation process creates a value named **CUSTOMERs** that correspond to the class collection for Customer. The xdrDefinition tabbed page in the Preview page of the class displays the following code:

```
<ElementType name = "CUSTOMERs" content = "eltOnly" order =
    "seq">
  <element type = "CUSTOMER"  minOccurs="0" maxOccurs="*"/>
</ElementType>
```

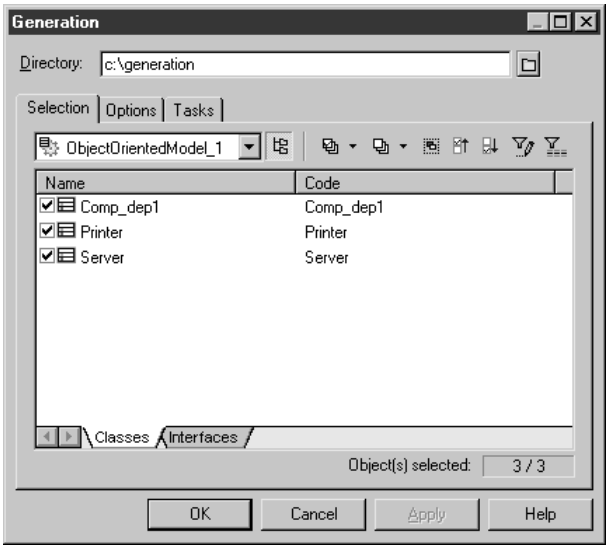
Generating XML files

The files generated are generated with the .xsd, .dtd or .xdr extension depending on the XML format currently used. A generation log file is also created after generation.

When you generate XML from an OOM, PowerDesigner creates one XML file per package. The file contains the definition of each of the classes you select to generate in the Generation dialog box. You can select any of the classes from the model, including those that are contained within packages or sub-packages.

❖ **To generate XML files:**

1 Select Language→Generate XML-*language* Code to display the Generation dialog box.



2 Type a destination directory for the generated file in the Directory box.
or
Click the Select a Path button to the right of the Directory box and browse to select a directory path.

3 Select the objects to include in the generation from the tabbed pages at the bottom of the Selection page.

4 Click the Options tab to display the Options page.

5 <optional> Select the Check Model check box if you want to verify the validity of your model before generation.

6 Select a value for each required option.

7 Click the Tasks tab to display the Tasks page.

8 Select the required task(s).

9 Click OK to generate.

A Progress box appears. The Result list displays the files that you can edit. The result is also displayed in the Generation page of the Output window, located in the bottom part of the main window.

XML files are generated in the destination directory.

CHAPTER 16

Generating from an Object-Oriented Model

About this chapter

This chapter describes how to generate an OOM, a CDM, and a PDM from an Object-Oriented Model.

Contents

Topic	Page
Generation basics	652
Generating an Object-Oriented Model from an Object-Oriented Model	655
Generating a Conceptual Data Model from an Object-Oriented Model	662
Generating a Physical Data Model from an Object-Oriented Model	675

Generation basics

When you generate from an Object-Oriented Model you can generate to an OOM, a CDM, or a PDM. You can generate a model from a global OOM or from a package within the model. Limiting model generation to a single package is useful when different designers own packages of the same OOM. Designers can generate their packages independently from others. Generating a package results in an independent model.

You can generate from an OOM in two ways:

Generate	Description
New model	Creates a new model containing the objects translated from the OOM
Updated model	Creates a default model with the objects translated from the OOM that is merged with an existing model. You can update, delete, or add objects in the existing model (model to be merged in the right pane) based on modifications made in the default model (in the left pane)

For more information on merging two models, see chapter Comparing and Merging Models in the *General Features Guide*.

Target models parameters

An external shortcut depends on a target object in a different model. External shortcuts allow you to share objects between different models.

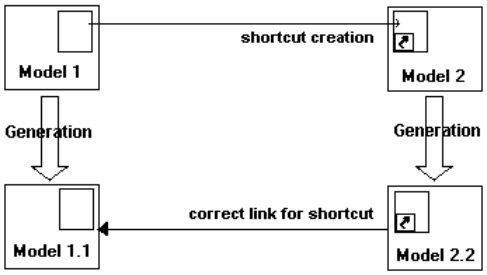
When you generate models into other models, you can preserve the link between an external shortcut and its target model through generation.

You can use the Target Models tab to select generated models to retrieve the corresponding target objects of external shortcuts. External shortcuts can then be correctly linked to target objects in the correct target model.

The Target Models page displays the following columns:

Column	Description
Target Models	Original target model of the shortcut (not editable)
Generated Models	Lets you select the model that will be used as the target for the generated shortcut

The model generation process allows you to define the target object of a shortcut in a generated model.



For example, here is the proper sequence of events for external shortcuts generation:

- Model 1 is the target model of a shortcut in Model 2.
- Model 1 is generated to Model 1.1.
- Model 2 is prepared for generation to Model 2.2 by associating appropriate parameters in the Target Models page:

  If Model 1 is still opened in the workspace The Target Model column displays the original target model (Model 1), and its path. The Generated Models column displays the last generated model the first time you generate Model 1; the next time you generate Model 1, the Generated Models column displays the last model selected. You can click the arrow in the Generated Models column to modify the model selection in order to allow the creation of a correctly linked shortcut.

  If Model 1 is closed in the workspace The Target Model column displays the original target model (Model 1), and its path. The Generated Models column displays <none>. When you click into the Generated Models column, the original target model Model 1 is automatically opened in the workspace in order to find the models generated from Model 1. You can use the arrow to select Model 1.1, the new target that will allow the creation of a correctly linked shortcut in Model 2.2.

- The external shortcut in Model 2.2 is correctly generated with a link to its target object in Model 1.1.

For more information on shortcuts, see chapter Managing Shortcuts in the *General Features Guide*.

Object selection parameters

You select objects for model generation from the Selection page.

Listing objects contained in a model or package

You can display in the list, objects in the current model, or objects in individual packages contained in the model. If you select the Include Sub-Objects tool, you can display either all objects in the current model, or all objects in a package.

You have the following selection options:

Parent object	Include Sub-objects selected	Include Sub-objects not selected
Model	All objects in the model including all objects in packages and sub-objects	All objects in the model except objects in packages and sub-objects
Package	All objects in the package including all sub-objects	All objects in the package except sub-objects

Objects selected in the model
Objects selected in the model can be automatically selected for generation by clicking the Use Graphical Selection tool in the Selection page tool bar.

Generating an Object-Oriented Model from an Object-Oriented Model

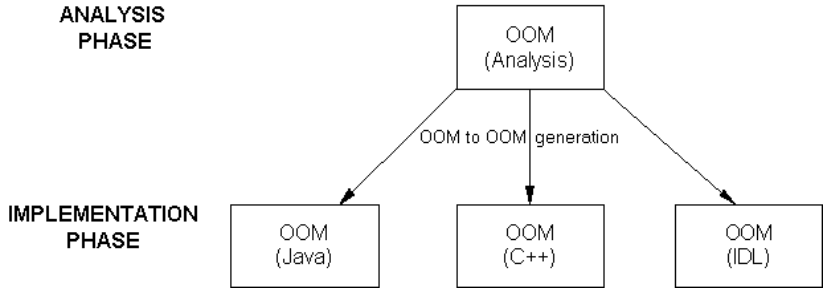
This section explains how to generate an Object-Oriented Model from an Object-Oriented Model.

Why generating an OOM into an OOM?

You can generate an OOM into an OOM when you need to create versions of a model. This kind of generation allows you to create a copy of a given model and define generation links between objects in the source OOM and their equivalent in the generated OOM. When changes are made to the source model, they can then be easily propagated to the generated models using the Update Existing Model generation mode.

Implementing for different languages

You can use the OOM to OOM generation to generate an analytical OOM (designed with the Analysis object language) into implementation OOMs designed for the different object languages supported in PowerDesigner.



Generating and updating an OOM

To generate an OOM, you must choose between one of the following methods:

- Generate new Object-Oriented Model
- Update existing Object-Oriented Model

Generate new Object-Oriented Model

You must indicate the following parameters when you generate a new OOM:

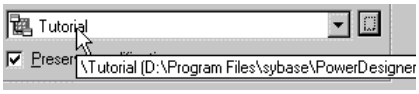
Parameter	Description
Object Language	Target object language
Share	Object language for the resulting OOM uses a shared object language stored in the Object language library
Copy	Object language for the resulting OOM uses a copy of the object language stored in the Object language library
Name	File name for the resulting OOM
Code	Reference code for the resulting OOM
Configure Model Options	Lets you define the model options for the new OOM

For more information on model options when generating an OOM from an OOM, see section Defining OOM model options in chapter Object-Oriented Model Basics.

Update existing Object-Oriented Model

You can generate an OOM into an existing OOM.

If you choose to update an existing OOM when the current OOM has never been generated, the Select Model dropdown listbox is empty by default. You have to click the Ellipsis button to display a Select Model dialog box in which you can select a model. The Workspace Location and Physical Path columns provide more information about the model location in the workspace and the file path to help you choose the OOM. You can also use the tool tip in the dropdown listbox to verify the location and path of the model.



To update an existing OOM, you must indicate the following parameters:

Parameter	Description
Select Model	Existing OOM. The OOM is generated into an OOM. This OOM is merged with the existing OOM to create an updated OOM. The dropdown listbox displays already generated models. The Ellipsis button lets you select models opened in the workspace
Preserve Modifications	Allows a comparison and merge of the newly generated OOM (default OOM) with the currently selected OOM

Clearing the Preserve Modifications check box
If the Preserve Modifications check box is not selected, PowerDesigner automatically replaces the existing OOM with the newly generated OOM. If you want to choose which objects to add or delete from the existing OOM, you must select Preserve Modifications to compare and merge the two OOM.

Defining OOM generation options

You can set the following general generation options from the Detail page:

Option	Description
Check Model	When selected, verifies the model before generating the OOM, and stops generation if an error is found
Save Generation Dependencies	When selected, PowerDesigner keeps track of the identity of each generated object. It is useful when merging two OOM which have been generated from the same OOM. Objects can be compared and recognized as the same object, even if the object has been modified in the merged OOM

Check model before generation
If you select the Check Model option, the procedure to generate an OOM starts by checking the validity of the OOM. An OOM results when no errors are found. You can set check options by selecting Tools→Check Model.

Generating a new OOM from an OOM

You can generate an OOM from an OOM or from a package in an OOM. PowerDesigner creates a new OOM containing all the objects that you selected to generate in the OOM. The newly created OOM appears in the browser and the corresponding diagram opens in the work area.

You can only generate an OOM from the active OOM diagram.

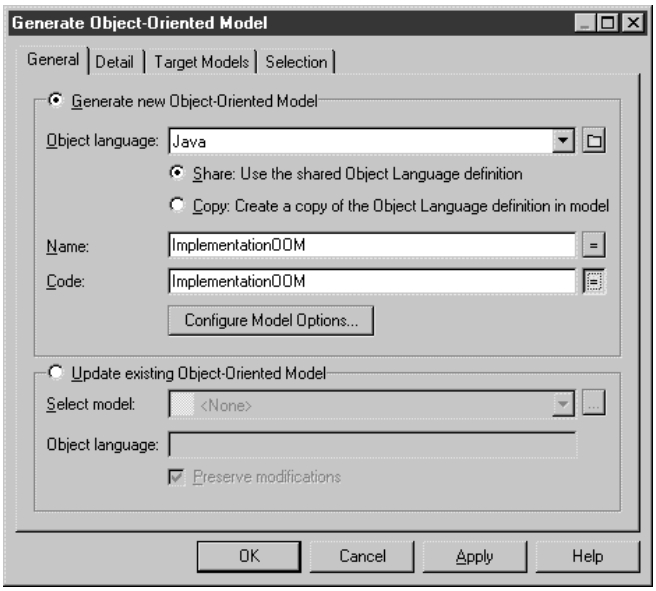
❖ **To generate a new OOM from an OOM:**

1 Select Tools→Generate Object-Oriented Model to display the Generate Object-Oriented Model dialog box.

2 Select the Generate New Object-Oriented Model radio button.

3 Select an object language in the Object Language dropdown listbox.

4 Type a new name and code, otherwise the OOM will have the same name and code as the source OOM.



5 <optional> Click Configure Model Options to define the options of the generated PDM.

6 Click the Detail tab to define options and generation parameters.

7 Click the Target Models tab to select the target models of shortcuts in the current model.

8 Click the Selection tab to display the Selection page.

9 Select the name of an OOM from the Select Location dropdown listbox.

> **Generating an OOM from a package**
> To generate an OOM from a package, select the package name from the dropdown listbox in the upper left corner of the dialog box. To generate an OOM from a sub-package, select a sub-package from the dropdown listbox in the upper left corner of the dialog box, or select a package name and click the Include Sub-Objects tool next to this dropdown listbox.

10 Select the check boxes for the objects you want to generate, and clear the check boxes for the objects you do not want to generate.

11 Click OK.

The Output window shows the progress of the generation process. The diagram of the new OOM appears in the work area.

Updating an existing OOM

There are two ways to update an existing OOM depending on whether the Preserve Modifications option is selected or not:

Preserve Modifications	Result
Selected	You can manually compare and merge an existing OOM (right pane) with the newly generated OOM (left pane)
Not selected	The existing OOM is automatically replaced by the newly generated OOM

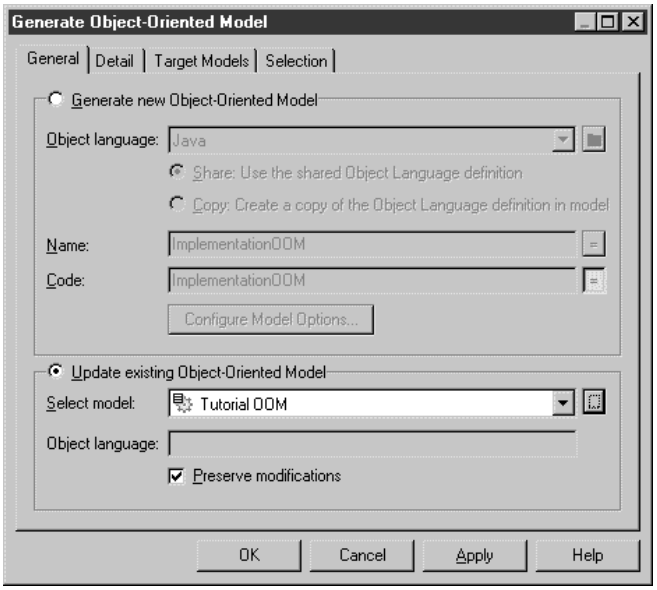
You can only generate an OOM from the active OOM diagram window.

❖ **To update an existing OOM by generating from an OOM:**

1 Select Tools→Generate Object-Oriented Model to display the Generate Object-Oriented Model dialog box.

2 Select the Update Existing Object-Oriented Model radio button.

3 Select a model from the Select Model dropdown listbox if the current model has already been generated.
or
Click the Ellipsis button beside the Select Model dropdown listbox and select an existing model in the Select a Model dialog box.



Preserve modifications
If you want to preserve objects in the OOM, then the Preserve Modification check box must be selected. If you clear this check box, all existing objects in the OOM will be removed from the model, leaving only the objects generated from the OOM.

4 Click the Detail tab to define options and generation parameters.

5 Click the Target Models tab to select the target models of shortcuts in the current model.

6 Click the Selection tab to display the Selection page.

7 Select the name of an OOM from the Select Location dropdown listbox. The default PDM is generated from this PDM.

Generating an OOM from a package
To generate an OOM from a package, select the package name from the dropdown listbox in the upper left corner of the dialog box. To generate an OOM from a sub-package, select a sub-package from the dropdown listbox in the upper left corner of the dialog box, or select a package name and click the Include Sub-Objects tool next to this dropdown listbox.

8 Select the check boxes for the objects you want to generate, and clear the check boxes for the objects you do not want to generate.

9 Click OK.

If you selected the Preserve Modifications check box, the Merge Models window appears.

If you cleared the Preserve Modifications check box, the updated OOM diagram appears in the work area.

Merging models
The Merge Models dialog box shows the newly generated OOM in the left pane, and the existing OOM in the right pane. You can select or clear object check boxes in the right pane for OOM objects that you want to include or delete in the model to be merged.

For more information on merging models, see chapter Comparing and Merging Models in the *General Features Guide*.

Generating a Conceptual Data Model from an Object-Oriented Model

This section explains how to generate a Conceptual Data Model from an Object-Oriented Model.

Generating CDM objects

If you have been working on a class diagram in an Object-Oriented Model (OOM) and want to change approach, you can generate a Conceptual Data Model (CDM) of an OOM class diagram. This process translates OOM objects into CDM objects and allows you to switch to an entity/relationships or Merise analysis.

You will then be able to further design your model and eventually generate a Physical Data Model (PDM) from the CDM.

When you generate a CDM from an OOM, PowerDesigner translates OOM objects and data types to CDM objects and data types.

The current object language of an OOM has no effect on generation to a CDM.

CDM generation translates OOM objects into conceptual objects.

OOM object	CDM object after generation
Domain	Domain
Class	Entity
Interface	Not translated
Attribute	Attribute
Identifier	Identifier
Operation	Not translated
Association	Relationship or association
Association class	Entity/relationship notation: the association class becomes an entity with two associations. Merise notation: the class becomes an association, and the OOM associations will be generated as association links
Dependency	Not translated
Realization	Not translated
Generalization	Inheritance

Translating OOM to CDM data types

PowerDesigner supports object languages and conceptual data types. When you generate objects from an OOM to a CDM, the data types of the selected object language are translated into conceptual data types. PowerDesigner conceptual data types cannot be modified.

You can check the mappings between object language and conceptual data types from the object language, you have to expand the Settings category and click the DataTypes category to display the following values:

Value	Description
BasicDataTypes	List of data types of the current object language and their equivalent conceptual data types. BasicDataTypes are the most common data types used
ConceptualDataTypes	List of conceptual data types with their equivalent data type in the current object language. They are the internal data types of PowerDesigner, and cannot be modified
AdditionalDataTypes	List of additional data types. Can be used to add or change data types of your own
DefaultDataTypes	Default data types used in an OOM

For more information on data types, see section Data types category, in chapter Object Languages Reference Guide in the *Advanced User Documentation.*

Managing object persistence during OOM to CDM generation

Developers tend to use object-oriented programming languages like Java, to develop business objects and components. Eventually, these objects can be represented in a conceptual data model prior to being stored in a database. A problem arises because object codes in object-oriented programming languages are often different than codes used in a CDM.

You can define persistent codes to bypass this impedance-mismatch.

Defining object persistence in the OOM

Persistent codes are codes defined for OOM classes and attributes that are used during OOM to CDM generation. They also facilitate round-trip engineering by allowing to recover object codes from the conceptual data model.

You can also define persistent data types for class attributes and domains. For data type persistence management, you have to take into account the following parameters:

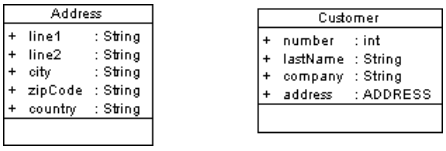
- The data type is **simple**, based on standard OOM data types. If the data type is persistent, generation keeps persistent data type in the target model. Otherwise data type is converted to standard data type in the target model
- The data type is **complex**, based on a classifier. If the data type is persistent, you can define different generation options according to what you want to have in the target model and to the attribute multiplicity

For more information on the generation of complex abstract data types, see section Managing complex data type persistence.

Managing complex data type persistence

In the following example, both classes are defined as persistent.

Class Customer contains an attribute address with a complex data type ADDRESS. This data type is based on the code of class Address.



For more information on how to use a class as a data type, see section Attaching a classifier to a data type or a return type in chapter Building a class diagram.

When you apply a persistent complex data type to a class attribute or a domain, you can define the following different generation options:

- Standard data type generation
- Embedded data type generation

Generating as persistent class

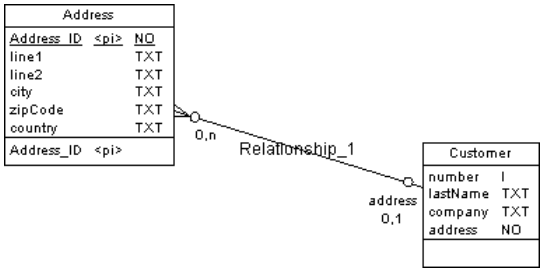
In the Detail page of the property sheet of an attribute using a complex data type, you have to select:

- The Persistent check box
- Persistent in the Class Generation dropdown listbox

After OOM to CDM generation, both classes become entities and a relationship is created between them. The class used as data type contains a new primary identifier attribute.

Example

If you define the standard data type generation options in the Address/Customer example, the generated CDM is the following:



The new column Address_ID is the primary identifier in entity Address.

Generating as embedded class

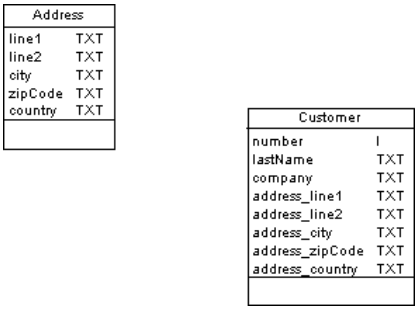
In the Detail page of the property sheet of an attribute using a complex data type, you have to select:

- The Persistent check box
- Embedded in the Class Generation dropdown listbox

After OOM to CDM generation, all the attributes of the class used as data type are embedded into the entity containing the attribute with complex data type. These attributes are prefixed by the name of the attribute using the complex data type.

Example

If you define the embedded data type generation options in the Address/Customer example, the generated CDM is the following:

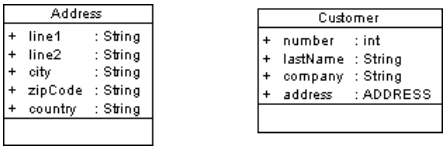


All the attributes of class Address are embedded into entity Customer and prefixed by "address" which is the name of the attribute using Address as complex data type.

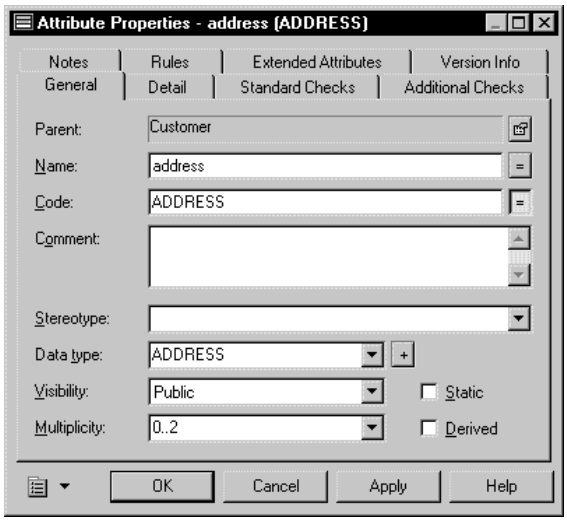
Managing multiplicity for complex persistent data types

If you define a multiplicity on a class attribute with a complex data type, the generation will reflect this multiplicity.

In the following example, attribute address in class Customer has a complex data type based on class Address.

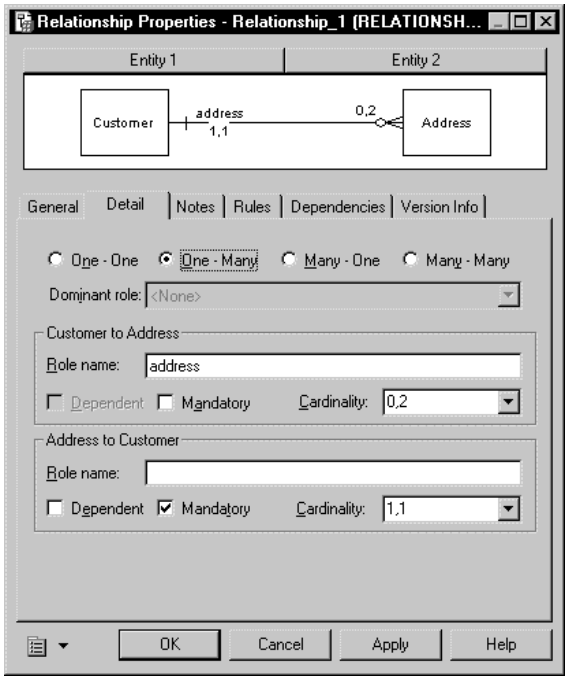


You also define the multiplicity 0..2 for attribute address:



Generating as persistent class

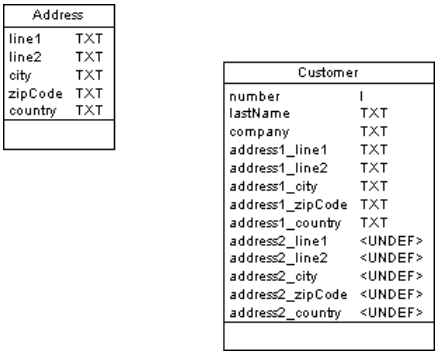
When the Persistent check box is selected for the attribute and the class generation is also set to Persistent, then the attribute multiplicity becomes a cardinality on the relationship between entities after OOM to CDM generation.



Generating as embedded class

When the Persistent check box is selected for the attribute and the class generation is set to Embedded, all the attributes of the class used as data type are embedded into the entity containing the attribute with complex data type. Attributes are generated as many times as required by multiplicity, provided multiplicity indicates a fixed value.

In the following example, attribute multiplicity is 0..2, so attributes will be embedded twice:



Embedded generation of complex data types is not possible when multiplicity is undefined (0..n).

Generating and updating a CDM

To generate a Conceptual Data Model, you have two possibilities:

- Generate a new Conceptual Data Model
- Update an existing Conceptual Data Model

Generate new Conceptual Data Model

You must indicate the parameters below when you generate a new CDM:

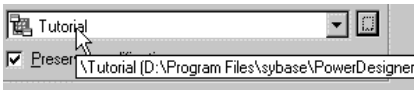
Parameter	Description
Name	File name for the resulting CDM
Code	Reference code for the resulting CDM
Configure Model Options	Lets you define the model options for the resulting CDM

For more information on configuration of the generated model options, see section Configuring the generated model options.

Update existing Conceptual Data Model

You can generate an OOM into an existing CDM.

If you choose to update an existing CDM when the current OOM has never been generated, the Select Model dropdown listbox is empty by default. You have to click the Ellipsis button to display a Select Model dialog box in which you can select a model. The Workspace Location and Physical Path columns provide more information about the model location in the workspace and the file path to help you choose the CDM. You can also use the tool tip in the dropdown listbox to verify the location and path of the model.



To update an existing CDM, you must indicate the following parameters:

Parameter	Description
Select Model	Existing CDM. The OOM is generated into a CDM. This CDM is merged with the existing CDM to create an updated CDM. The dropdown listbox displays already generated models. The Ellipsis button lets you select models opened in the workspace
Preserve Modifications	Allows a comparison and merge of the newly generated CDM (default CDM) with the currently selected CDM

Clearing the Preserve Modifications check box
If the Preserve Modifications check box is not selected, PowerDesigner automatically replaces the existing CDM with the newly generated CDM. If you want to choose which objects to add or delete from the existing CDM, you must select Preserve Modifications to compare and merge the two CDM.

Configuring the generated model options

When you configure the options of a CDM to generate, you may define options diverging from the OOM options.

To avoid conflicts, PowerDesigner applies the following rule for default values of CDM options: an option defined for the generated CDM should respect the equivalent option of the OOM.

Equivalent options are model options found in both models. The Enforce non-divergence option exists in both OOM and CDM.

OOM option	CDM option	Result in generated CDM
✓ Enforce non-divergence	_	Enforce non-divergence in model according to OOM options. Data items and attributes attached to the domain cannot have divergent definitions
_	✓ Enforce non-divergence	Enforce non-divergence in model according to CDM options defined using the Configure Model Options feature.

Relationships and data items unique code

Unique Code for relationships and data item has no equivalent in the OOM and this option is not selected by default in the CDM options. However, if you select Unique Code for relationships or data items in the CDM options, relationships and data items will be renamed during the generation of an OOM to a CDM.

Options with no equivalent, like Show Classes as Data Type in the OOM without any corresponding option in a CDM, are generated using default values found in the registry.

Defining CDM generation options

You can set the following general generation options:

Option	Description
Check model	When selected, checks the model before generating the CDM and stops generation if an error is found
Save generation dependencies	When selected, PowerDesigner keeps track of each generated object identity. It is useful when merging two CDM which have been generated from the same OOM. Objects can be compared and recognized as the same object, even if the object has been modified in the merged CDM
Convert Names into Codes	When selected, object codes are generated from names using the corresponding conversion script. This is useful when generating models with very different naming conventions. For example a Java class attribute code may be "customerName" whereas a CDM entity attribute may be "CUSTOMER_NAME". With this option selected, both objects will have their codes generated from their names. If you do not select this option, generated object codes will be copied from original object codes

For more information on conversion scripts, see section Using a conversion script, in chapter Managing models in the *General Features Guide*.

Check model before generation
If you select the Check Model option, the procedure to generate a CDM starts by checking the validity of the OOM. A CDM results when no errors are found. You can set check options by selecting Tools→Check Model.

Generating a new CDM from an OOM

When you generate from a class diagram of an OOM to a new CDM, PowerDesigner creates a new CDM with all the objects that you selected to generate in the OOM. The newly created CDM appears in the browser and the corresponding diagram opens in the diagram window.

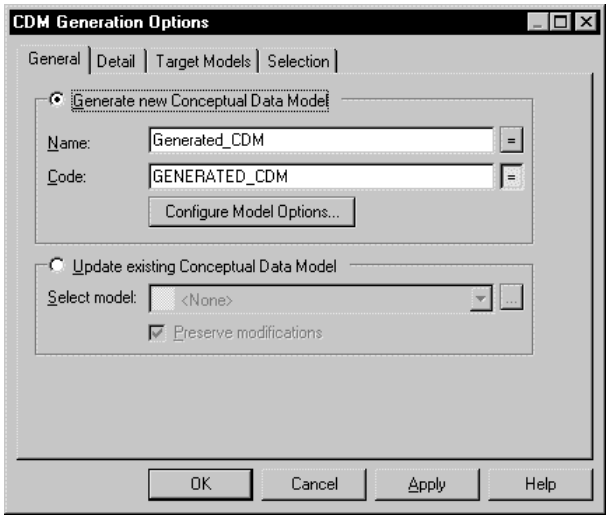
You can only generate a CDM from the active OOM diagram window.

❖ **To generate to a new CDM from an OOM:**

1 Select Tools→Generate Conceptual Data Model to display the CDM Generation Options dialog box.

2 Click the Generate New Conceptual Data Model radio button.

3 Type a new name and code, otherwise the CDM will have the same name and code as the OOM.



4 <optional> Click the Configure Model Options to define the options of the generated CDM.

5 Click the Detail tab to define options and generation parameters.

6 Click the Target Models tab to select the target models of shortcuts in the current model.

7 Click the Selection tab to display the Selection page.

8 Select the name of an OOM from the Select Location dropdown list box.

> **Generating a CDM from a package**
> To generate a CDM from a package, select the package name from the dropdown listbox in the upper left corner of the dialog box.
>
> To generate a CDM from a sub-package, select a sub-package from the dropdown listbox in the upper left corner of the dialog box, or select a package name and click the Include Sub-Objects tool next to this dropdown listbox.

9 Select the check boxes for the objects you want to generate, and clear the check boxes for the objects you do not want to generate.

10 Click OK.

The Generation page in the Output window, located in the lower part of the main window, shows the progress of the generation process. The new CDM appears in the diagram window.

Updating an existing CDM

There are two ways to update an existing CDM depending on whether the Preserve Modifications option is selected or not:

Preserve Modifications	Result
Selected	You can manually compare and merge an existing CDM (right pane) with the newly generated CDM (left pane)
Not selected	The existing CDM is automatically replaced by the newly generated CDM

If the Preserve Modifications check box is selected, the Merge Models window appears after the new CDM has been successfully generated. You can use the Merge window to select objects to be updated, deleted, or added to the model to be merged.

The model must be open in the workspace to be merged with a source model.

If the Preserve Modifications check box is not selected, the existing model is replaced.

You can only generate a CDM from the active OOM diagram window.

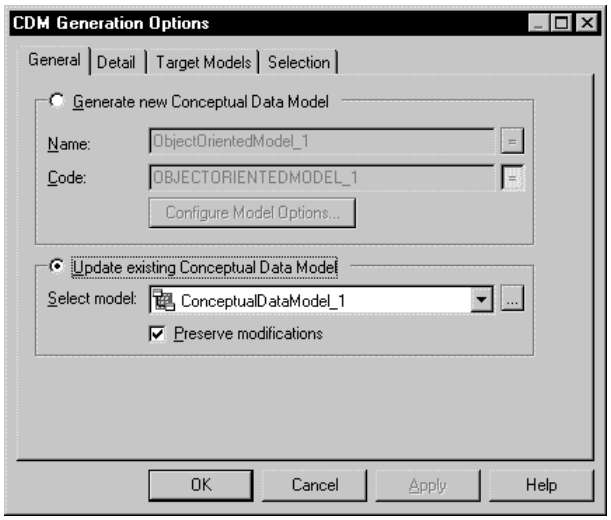
❖ **To update an existing CDM by generating from an OOM:**

1 Select Tools→Generate Conceptual Data Model to display the CDM Generation Options dialog box.

If you do not have a CDM in the current workspace, the Update Existing Conceptual Data Model option is not available.

2 Select the Update Existing Conceptual Data Model radio button.

3 Select a model from the Select Model dropdown listbox if the current model has already been generated.
or
Click the Ellipsis button beside the Select Model dropdown listbox and select and existing model in the Select a Model dialog box.



Preserve modifications
If you want to preserve the existing objects in the CDM, then the Preserve modifications check box must be selected. If you clear this check box, all existing objects in the CDM will be removed from the model, leaving only the objects generated from the OOM.

4 Click the Detail tab to define options and generation parameters.

5 Click the Target Models tab to select the target models of shortcuts in the current model.

6 Click the Selection tab to display the Selection page.

7 Select the name of an OOM from the Select Location dropdown list box. The default CDM is generated from this OOM.

> **Generating a CDM from a package**
> To generate a CDM from a package, select the package name from the dropdown listbox in the upper left corner of the dialog box.
>
> To generate a CDM from a sub-package, select a sub-package from the dropdown listbox in the upper left corner of the dialog box, or select a package name and click the Include Sub-Objects tool next to this dropdown listbox.

8 Select the check boxes for the objects you want to generate, and clear the check boxes for the objects you do not want to generate.

9 Click OK.

If you selected the Preserve Modifications check box, the Merge Models window appears.

If you cleared the Preserve Modifications check box, the updated CDM appears in the diagram window.

> **Merging models**
> The Merge Models dialog box shows the newly generated CDM in the left pane and the existing CDM in the right pane. You can select or clear object check boxes in the right pane for CDM objects you want to include or delete in the model to be merged.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

Generating a Physical Data Model from an Object-Oriented Model

You may want to go from a class diagram of an Object-Oriented Model (OOM) to a Physical Data Model (PDM) because the design of your system in the OOM can be directly used by the database you want to work with. The advantage of this process is to model the objects in the world they live in and, through the generation, to automate the translation to tables and fields in the database.

When you generate a PDM from an OOM, PowerDesigner translates OOM objects and data types to PDM objects and data types supported by the current Database Management System (DBMS). The current object language of an OOM has no effect on the generation to a PDM.

Generating PDM objects

PDM generation translates OOM objects into physical objects.

OOM object	PDM object after generation
Domain	Domain
Class	Table (only if the Persistent and Generate check boxes are selected in the class property sheet)
Interface	Not translated
Attribute	Column (only if the Persistent check box is selected in the attribute property sheet)
Identifier	Identifier
Operation with <<storedProcedure>> stereotype	If the parent class is generated as a table, the operation is generated as a stored procedure attached to the table. The operation body is generated as a body in the procedure definition. The generated procedure has the same name as the operation, however, if a name conflict occurs, a number is added to the name of the procedure

OOM object	PDM object after generation
Operation with <<storedFunction>> stereotype	If the parent class is generated as a table, the operation is generated as a stored function attached to the table. The operation body is generated as a body in the function definition. The generated function has the same name as the operation, however, if a name conflict occurs, a number is added to the name of the function
Association	Reference or table
Association class	A table and two associations between the end points of the association class
Dependency	Not translated
Realization	Not translated
Generalization	Reference

Generating from classes

For a class to become a table, the Persistent and Generate check boxes must be selected in the class property sheet.

The cardinality of a class becomes the number of records of a table.

Generating from associations

If the association has a many-to-many multiplicity: both roles of the association have the * sign selected in their multiplicity dropdown listboxes, the association is translated into a table in the generated PDM. If it has any other multiplicity, the association becomes a reference.

A role name becomes a migrated foreign key after PDM generation.

Translating OOM to PDM data types

PowerDesigner supports both object languages and physical data types. Data types you select in the OOM are not always supported by the current DBMS. In this case, the data type is translated to a data type supported by the DBMS when you generate the PDM. Physical data types are converted into conceptual data types (the internal types in PowerDesigner).

You can check the mappings between object language and physical data types in the object language, from Language→Edit Current Object Language. The physical data types are converted into conceptual data types.

In the object language, you have to expand the Settings category and click the DataTypes category to display the following values:

Value	Description
BasicDataTypes	List of data types of the current object language and their equivalent conceptual data types. BasicDataTypes are the most common data types used
AdditionalDataTypes	List of additional data types. Can be used to add or change data types of your own
ConceptualDataTypes	List of conceptual data types with their equivalent data type in the current object language. They are the internal data types of PowerDesigner, and cannot be modified
DefaultDataTypes	Default data types used in an OOM

For more information on data types, see section Data types category, in chapter Object Languages Reference Guide in the *Advanced User Documentation.*

Managing object persistence during OOM to PDM generation

Developers tend to use object-oriented programming languages like Java, to develop business objects and components. Eventually, these objects need to be stored in a database, then a problem arises when the user tries to store objects in a relational database because object codes in object-oriented programming languages are often different than codes used in a relational database.

You have to modify object codes after generating an OOM into a PDM to be compliant with the DBMS.

You can define persistent codes to bypass this impedance-mismatch.

You can also create object to relational mappings between OOM and PDM objects to further define the links between objects from different models.

For more information on O/R Mapping, see section Understanding O/R mapping, in chapter Working with Object-Oriented Models.

Defining object persistence in the OOM

Persistent codes are defined for OOM classes and attributes. They are used during OOM to PDM generation in order to fine tune data storage in a relational database. They also facilitate round-trip engineering by allowing to recover object codes from the database.

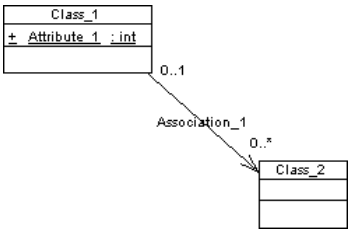
You can also define persistent data types for class attributes and domains. For data type persistence management, you have to take into account the following parameters:

- The data type is **simple**, based on standard OOM data types. If the data type is persistent, generation keeps persistent data type in the target model. Otherwise data type is converted to standard data type in the target model
- The data type is **complex**, based on a classifier. If the data type is persistent, you can define different generation options according to what you want to have in the target model and to the attribute multiplicity

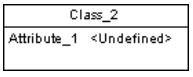
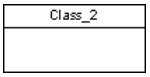
For more information on the generation of complex abstract data types, see section Managing complex data type persistence.

Attribute migration

If you generate the following OOM into a PDM:

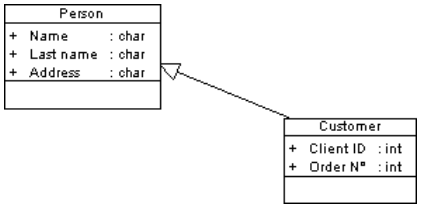


The migration of attributes in the generated PDM depends on the persistence and generation options of the classes in the source OOM:

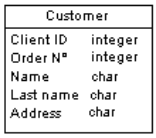
Class 1	Class 2	Result
Generate not selected Persistence selected	Generate selected Persistence selected	
Generate not selected Persistence not selected	Generate selected Persistence selected	

Inherited attributes

In the following OOM, class Customer inherits from Person via a generalization link. Person is not generated but persistent, and Customer is generated and persistent.



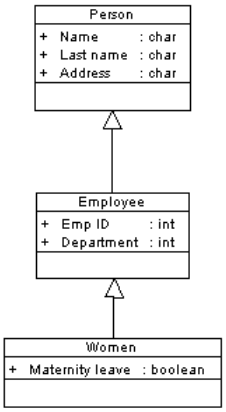
Since Person is persistent, Customer inherits the attributes of the parent class in the generated PDM:



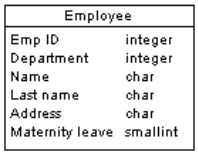
Derived class

A derived class is created for conceptual reasons to improve the readability of a model but it adds no semantic information. There is no point of generating a derived class in a PDM, and attributes of this class should be migrated to parent.

In the following example, class Women is created to further analyze class Employee and highlight a specific attribute "Maternity leave". This class is derived, thus not generated but persistent:



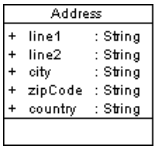
In the generated PDM, Employee inherits from parent class and also from derived class:

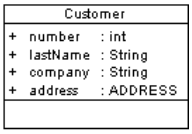


Managing complex data type persistence

In the following example, both classes are defined as persistent.

Class Customer contains an attribute address with a complex data type ADDRESS. This data type is based on the code of class Address.





For more information on how to use a class as a data type, see section Attaching a classifier to a data type or a return type in chapter Building a class diagram.

When you apply a persistent complex data type to a class attribute or a domain, you can define the following different generation options:

- Standard data type generation
- Embedded data type generation
- Abstract data type generation

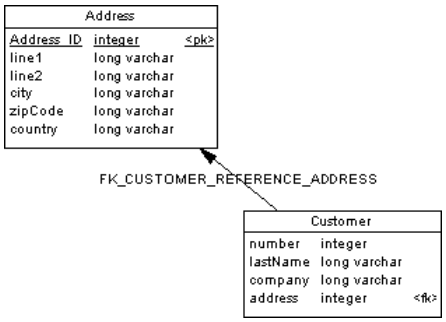
Generating as persistent class

In the Detail page of the property sheet of an attribute using a complex data type, you have to select:

- The Persistent check box
- Persistent in the Class Generation dropdown listbox

After OOM to PDM generation, both classes become tables and a reference is created between them. This reference contains a join linking the primary key column in the parent table Address (this column is new, it was created during generation) and the foreign key column in the child table Customer. The foreign key column stands for the attribute using a complex data type in the OOM.

For example, if you define the standard data type generation options in the Address/Customer example, the generated PDM is the following:



The reference contains a join between Address_ID (new column in table Address) and address in child table Customer. Address_ID contains the primary key migrated to column address.

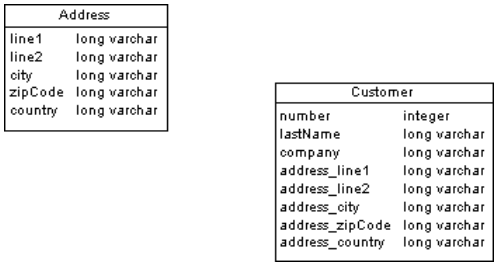
Generating as embedded class

In the Detail page of the property sheet of an attribute using a complex data type, you have to select:

- The Persistent check box
- Embedded in the Class Generation dropdown listbox

After OOM to PDM generation, all the attributes of the class used as data type are embedded into the table containing the attribute with complex data type. These attributes are prefixed by the name of the attribute using the complex data type.

For example, if you define the embedded data type generation options in the Address/Customer example, the generated PDM is the following:



All the attributes of class Address are embedded into table Customer and prefixed by "address" which is the name of the attribute using Address as complex data type.

Generating as abstract data type class

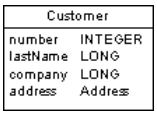
In the Detail page of the property sheet of an attribute using a complex data type, you have to select:

- The Persistent check box
- Embedded in the Class Generation dropdown listbox

You also have to select the Abstract check box in the Detail page of the property sheet of the class used as data type.

After OOM to PDM generation, the class used as data type becomes an abstract data type.

For example, if you define the abstract data type generation options in the Address/Customer example, the generated PDM is the following:

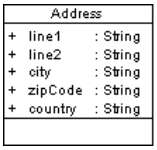


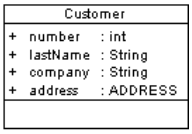
Class Address is generated as an abstract data type attached to column address in table Customer.

Managing multiplicity for complex persistent data types

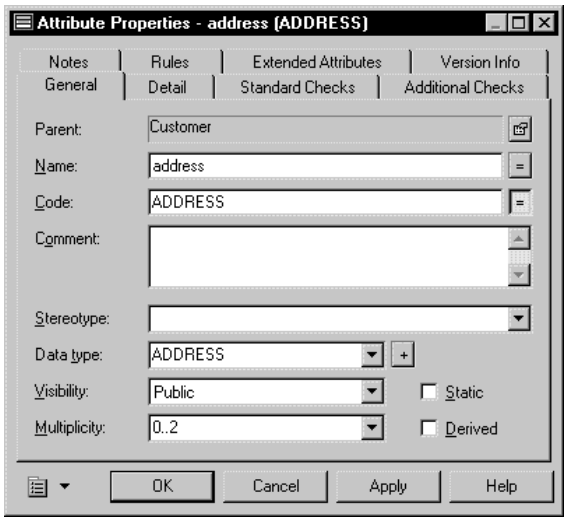
If you define a multiplicity on a class attribute with a complex data type, the generation will reflect this multiplicity.

In the following example, attribute address in class Customer has a complex data type based on class Address.



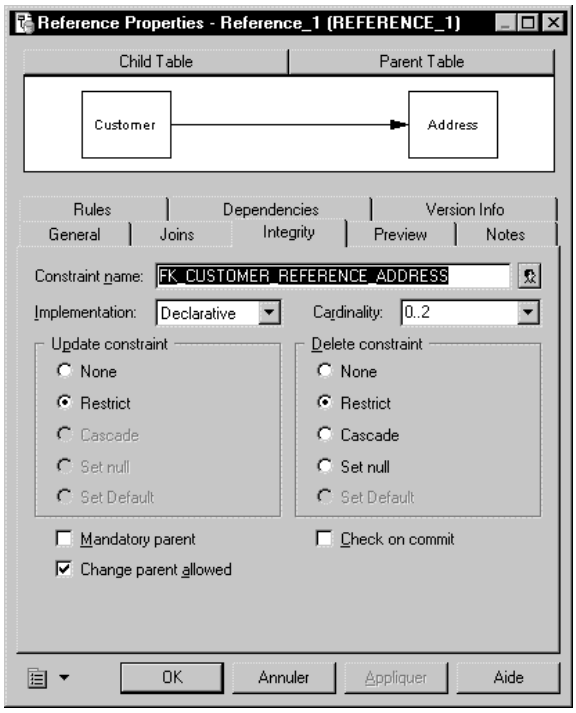


You also define the multiplicity 0..2 for attribute address:



Generating as persistent class

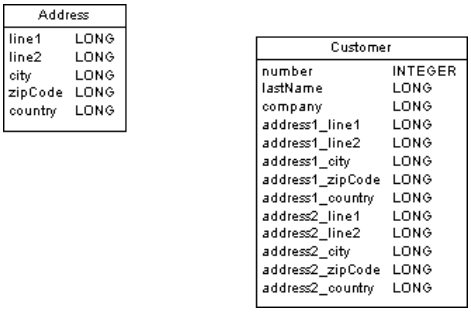
When the Persistent check box is selected for the attribute and the class generation is also set to Persistent, then the attribute multiplicity becomes a cardinality on the reference between tables after OOM to PDM generation.



Generating as embedded class

When the Persistent check box is selected for the attribute and the class generation is set to Embedded, all the attributes of the class used as data type are embedded into the table containing the column created for the attribute with complex data type. Attributes are generated as many times as required by multiplicity, provided multiplicity indicates a fixed value.

In the following example, attribute multiplicity is 0..2, so attributes will be embedded twice:



Embedded generation of complex data types is not possible when multiplicity is undefined (0..n).

Generating as abstract data type class

When both attribute and class generation are set to Persistent, and when the class used as data type is generated as an abstract data type, the class is generated as an abstract data type without taking into account the multiplicity of the generated attribute.

However, for those DBMS that support ARRAY and LIST for abstract data types, the generation of a class into an abstract data type can reflect the attribute multiplicity in the following way:

Multiplicity	Generated as
0..n or 1..n	Abstract data type with the LIST type (example: TABLE for Oracle)
1..2 or 0..10	Abstract data type with the ARRAY type (example: VARRAY for Oracle)
0..1 or 1..1	Abstract data type with the OBJECT type

Generating and updating a PDM

To generate a Physical Data Model, you have two possibilities:

- Generate a new Physical Data Model

- Update an existing Physical Data Model

Generate new Physical Data Model

You must indicate the parameters below when you generate a new PDM:

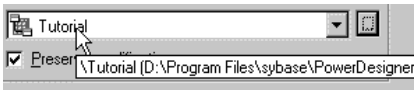
Parameter	Description
DBMS	Database Management System definition for the resulting PDM
Share	The resulting PDM uses a shared DBMS definition file stored in the DBMS library
Copy	The resulting PDM uses a copy of the DBMS definition file in the model
Name	File name for the resulting PDM
Code	Reference code for the resulting PDM
Configure Model Options	Lets you define the model options for the resulting PDM

For more information on configuration of the generated model options, see section Configuring the generated model options.

Update existing Physical Data Model

You can generate an OOM into an existing PDM.

If you choose to update an existing PDM when the current OOM has never been generated, the Select Model dropdown listbox is empty by default. You have to click the Ellipsis button to display a Select Model dialog box in which you can select a model. The Workspace Location and Physical Path columns provide more information about the model location in the workspace and the file path to help you choose the PDM. You can also use the tool tip in the dropdown listbox to verify the location and path of the model.



To update an existing PDM, you must indicate the following parameters:

Parameter	Description
Select Model	Existing PDM. The OOM is generated into a PDM. This PDM is merged with the existing PDM to create an updated PDM. The dropdown listbox displays already generated models. The Ellipsis button lets you select models opened in the workspace
DBMS	Current Database Management System definition for the existing PDM
Preserve Modifications	Allows a comparison and merge of the newly generated PDM (default PDM) with the currently selected PDM

Clearing the Preserve Modifications check box
If the Preserve Modifications check box is not selected, PowerDesigner automatically replaces the existing PDM with the newly generated PDM. If you want to choose which objects to add or delete from the model to be merged, you must select Preserve Modifications to compare and merge the two PDM.

Configuring the generated model options

When you configure the options of a PDM to generate, you may define options diverging from the OOM options.

To avoid conflicts, PowerDesigner applies the following rule for default values of PDM options: an option defined for the generated PDM should respect the equivalent option of the OOM.

Equivalent options are model options found in both models. The Enforce non-divergence option exists in both OOM and PDM.

OOM option	PDM option	Result in generated PDM
✔ Enforce non-divergence	_	Enforce non-divergence in model according to OOM options. Columns attached to the domain cannot have divergent definitions
_	✔ Enforce non-divergence	Enforce non-divergence in model according to PDM options defined using the Configure Model Options feature.

References unique code

Unique Code for references has no equivalent in the OOM and this option is not selected by default in the PDM options. However, if you select Unique Code for references in the PDM options, references will be renamed during the generation of an OOM to a PDM.

Options with no equivalent, like Show Classes as Data Type in the OOM without any corresponding option in a PDM, are generated using default values found in the registry.

Defining PDM generation options

You can set the following general generation options:

Option	Description
Check Model	When selected, verifies the model before generating the PDM, and stops generation if an error is found
Save Generation Dependencies	When selected, PowerDesigner keeps track of the identity of each generated object. This is useful when merging two PDM which have been generated from the same OOM. Objects can be compared and recognized as the same object, even if the object has been modified in the merged PDM
O/R Mapping	When selected, allows to define the generated PDM as the data source of the current OOM and to create object correspondence between OOM and PDM
Convert Names into Codes	When selected, object codes are generated from names using the corresponding conversion script. This is useful when generating models with very different naming conventions. For example a Java class attribute code may be "customerName" whereas a PDM table column code may be "CUSTOMER_NAME". With this option selected, both objects will have their codes generated from their names. If you do not select this option, generated object codes will be copied from original object codes
Rebuild Triggers	When selected, allows to rebuild triggers with preservation option, after the PDM generation. Rebuild takes place after merge if you are updating an existing PDM

For more information on O/R Mapping, see section Understanding O/R mapping, in chapter Working with Object-Oriented Models.

For more information on conversion scripts, see section Using a conversion script, in chapter Managing models in the *General Features Guide*.

Check model before generation
If you select the Check Model option, the procedure to generate a PDM starts by checking the validity of the OOM. A PDM results when no errors are found. You can set check options by selecting Tools→Check Model.

Table parameters

The following parameter defines the naming convention for tables:

Parameter	Description
Table prefix	Prefix for the names of tables generated from classes in the OOM

Reference parameters

The following parameters define default referential integrity constraints:

Parameter	Description
Update rule	Default update constraint for referential integrity
Delete rule	Default delete constraint for referential integrity

For a description of referential integrity, see the glossary in the *Physical Data Model User's Guide*.

Foreign key name

By default, the name of the migrated foreign keys generated in a PDM is identical to the name of the primary key. In case of column name conflict, the first 3 letters of the name of the parent table are added before the name of the migrated column.

You can modify this default behavior and use templates for naming the migrated foreign keys. In the Reference groupbox, you can select one of the predefined templates in the FK Column Name Template dropdown listbox. These templates are saved in the registry, you can modify them, but their default value is preserved.

You can also create your own foreign key naming templates by typing a template directly in the FK Column Name Template box using the following variables:

Variable	Value
%REFR%	Generated code of the reference
%REFNAME%	Reference name
%REFRCODE%	Reference code
%PARENT%	Generated code of the parent table
%PNAME%	Parent table name
%PCODE%	Parent table code
%COLUMN%	Generated code of the parent column
%COLNNAME%	Parent column name for the corresponding PK
%COLNCODE%	Parent column code for the corresponding PK

Variable	Value
%KEY%	Name of the key constraint attached to the reference
%CONSTNAME%	Name of the key constraint attached to the reference (same as %KEY%)
%KNAME%	Name of the key attached to the reference
%KCODE%	Code of the key attached to the reference
%PROLE%	Role of the class that generated the parent table, this variable proceeds from the object-oriented environment. If no role is defined on the association, %PROLE% takes the content of %PARENT% to avoid generating a column with no name

You can also use the default PowerDesigner formatting variables in the template as explained in section List of PowerDesigner formatting variables in chapter Variables in PowerDesigner in the PDM User's Guide.

Customized naming templates reappear in the generation dialog box the next time you open it, but are not saved in the registry thus not added to the dropdown list of predefined templates.

For example, if you define the template %PROLE%, the name of the foreign key will be made of the role beside the class that generated the parent table containing the primary key. If the parent table contains multiple keys, the foreign key names will be automatically renamed.

In the Reference groupbox, you can select the Always Use Template radio button to always use the default or customized template for naming FK columns, even if there is no conflict with an existing column.

Index parameters

The following parameters define naming conventions for indexes:

Parameter	Description
PK index names	Naming convention for primary keys, by default %TABLE%_PK
Key index names	Naming convention for alternate keys, by default %TABLE%_AK
FK index names	Naming convention for foreign keys, by default %REFR%_FK
FK threshold	Minimum number of estimated occurrences of an entity necessary to create an index on a foreign key

PK index name variables

You can use the following variables in the PK index names and Key Index Names fields:

Variable	Value
%TABLE%	Generated code of the table. This is the table code generated in the database. It may be truncated if the code contains characters not supported by the DBMS
%TNAME%	Table name
%TCODE%	Table code
%TLABL%	Table comment

FK index name variables

You can use the following variables in the FK index name field. The generated code of a variable is the code defined in the object property sheet, it may be truncated when generated if the code contains characters not supported by the DBMS

Variable	Value
%REFR%	Generated code of the reference
%PARENT%	Generated code of the parent table
%PNAME%	Parent table name
%PCODE%	Parent table code
%CHILD%	Generated code of the child
%CNAME%	Child table name
%CCODE%	Child table code
%PQUALIFIER%	Parent table qualifier
%CQUALIFIER%	Child table qualifier
%REFRNAME%	Reference name
%REFRCODE%	Reference code

Preserving a modified primary key
If you modify a primary key in a PDM, then regenerate the PDM from an OOM, the modified primary key is not preserved. If you want to preserve a modified primary key, you need to modify the identifier in the OOM before regeneration.

Generating a new PDM from an OOM

When you generate from a class diagram in an OOM to a new PDM, PowerDesigner creates a new PDM with all the objects that you selected to generate in the OOM. The newly created PDM appears in the browser and the corresponding diagram opens in the diagram window.

You can only generate a PDM from the active OOM diagram window.

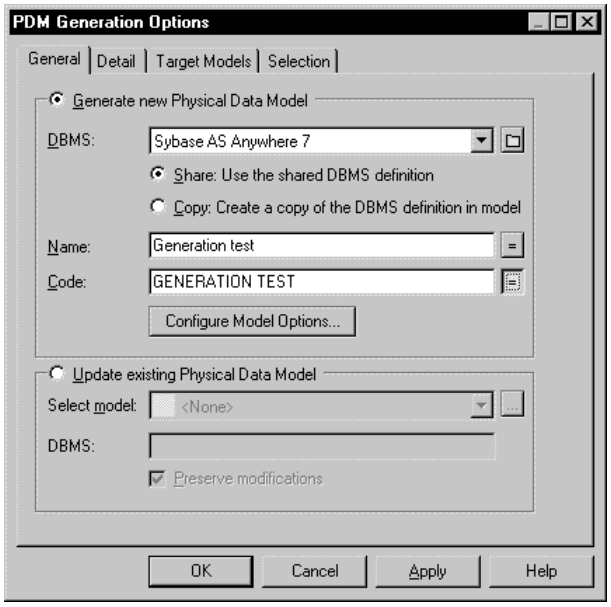
❖ To generate to a new PDM from an OOM:

1 Select Tools→Generate Physical Data Model to display the PDM Generation Options dialog box.

2 Click the Generate New Physical Data Model radio button.

3 Select the DBMS you want to associate to your model from the DBMS dropdown listbox.

4 Type a new name and code, otherwise the PDM will have the same name and code as the OOM.



5 <optional> Click the Configure Model Options to define the options of the generated PDM.

6 Click the Detail tab to define options and generation parameters.

7 Click the Target Models tab to select the target models of shortcuts in the current model.

8 Click the Selection tab to display the Selection page.

9 Select the name of an OOM from the Select Location dropdown listbox.

> **Generating a PDM from a package**
> To generate a PDM from a package, select the package name from the dropdown listbox in the upper left corner of the dialog box.
>
> To generate a PDM from a sub-package, select a sub-package from the dropdown listbox in the upper left corner of the dialog box, or select a package name and click the Include Sub-Objects tool next to this dropdown listbox.

10 Select the check boxes for the objects you want to generate, and clear the check boxes for the objects you do not want to generate.

11 Click OK.

The Generation page in the Output window shows the progress of the generation process. The new PDM appears in the diagram window.

Updating an existing PDM

There are two ways to update an existing PDM depending on whether the Preserve Modifications option is selected or not:

Preserve Modifications	Result
Selected	You can manually compare and merge existing PDM (right pane) with the newly generated PDM (left pane)
Not selected	The existing PDM is automatically replaced by the newly generated PDM

If the Preserve Modifications check box is selected, the Merge Models window appears after the new PDM has been successfully generated. You can use the Merge window to select objects to be updated, deleted, or added to the model to be merged. The model must be open in the workspace to be merged with a source model.

If the Preserve Modifications check box is not selected, the existing model is replaced.

You can only generate a PDM from the active OOM diagram window.

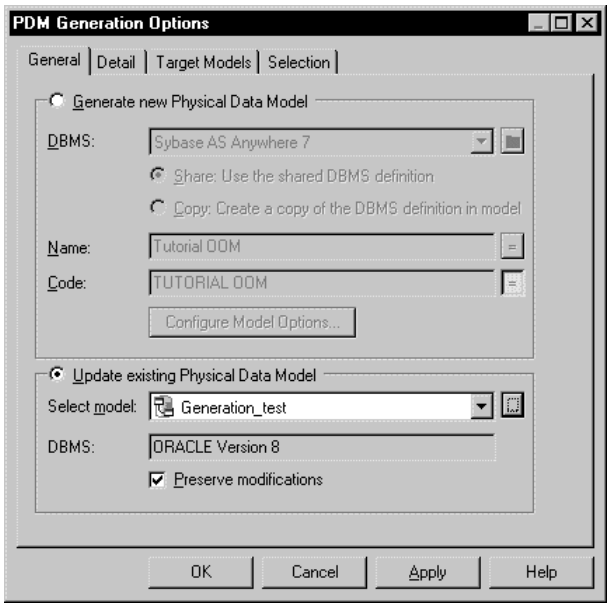
❖ **To update an existing PDM by generating from an OOM:**

1 Select Tools→Generate Physical Data Model.

If you do not have a PDM in the current workspace, the Update Existing Physical Data Model option is not available.

2 Select the Update Existing Physical Data Model radio button.

3 Select a model from the Select Model dropdown listbox if the current model has already been generated.
or
Click the Ellipsis button beside the Select Model dropdown listbox and select and existing model in the Select a Model dialog box.



The DBMS attached to the model appears in the DBMS box.

> **Preserve modifications**
> If you want to preserve the existing objects in the PDM, then the Preserve modifications check box must be selected. If you clear this check box, all existing objects in the PDM will be removed from the model, leaving only the objects generated from the OOM.

4 Click the Detail tab to define options and generation parameters.

5 Click the Target Models tab to select the target models of shortcuts in the current model.

6 Click the Selection tab to display the Selection page.

7 Select the name of an OOM from the Select Location dropdown listbox. The default PDM is generated from this OOM.

> **Generating a PDM from a package**
> To generate a PDM from a package, select the package name from the dropdown listbox in the upper left corner of the dialog box.
>
> To generate a PDM from a sub-package, select a sub-package from the dropdown listbox in the upper left corner of the dialog box, or select a package name and click the Include Sub-Objects tool next to this dropdown listbox.

8 Select the check boxes for the objects you want to generate, and clear the check boxes for the objects you do not want to generate.

9 Click OK.

If you selected the Preserve Modifications check box, the Merge Models window appears.

If you cleared the Preserve Modifications check box, the updated CDM appears in the diagram window.

> **Merging models**
> The Merge Models dialog box shows the newly generated PDM in the left pane, and the existing PDM in the right pane. You can select or clear object check boxes in the right pane for PDM objects that you want to include or delete in the model to be merged.

For more information on merging models, see section Merging Models, in chapter Comparing and Merging Models in the *General Features Guide*.

CHAPTER 17

Using Business Rules in an OOM

About this chapter

This chapter describes how business rules help you model information.

Contents

Topic	Page
What is a business rule in an OOM?	696
Defining business rules in an OOM	697
Applying business rules to OOM objects	700

What is a business rule in an OOM?

A business rule is a written statement specifying what the information system must do or how it must be structured to support business needs.

A business rule is a rule that your business follows. A business rule could be a government-imposed law, a customer requirement, or an internal guideline.

Starts as an observation

Business rules often start as simple observations, for example "customers call toll-free numbers to place orders." During the design process they develop into more detailed expressions, for example what information a customer supplies when placing an order or how much a customer can spend based on a credit limit.

Guides modeling

Business rules guide and document the creation of a model. For example, the rule "an employee belongs to only one division" can help you graphically build the link between an employee and a division.

Complements graphics

Business rules complement model graphics with information that is not easily represented graphically. For example, some rules specify physical concerns in the form of formulas and validation rules. These technical expressions do not have a graphical representation.

Check parameters

You can attach business rules to objects in an OOM. You can generate business validation rules as check parameters if the validation rules are attached to domains.

For more information on defining and using check parameters, see section Defining check parameters, in chapter Building a Class Diagram.

Defining business rules in an OOM

You can define a business rule which can be attached to OOM objects.

Business rule properties in an OOM

A business rule definition includes the following main properties:

Property	Description	Maximum length
Name	Name for the rule	254
Code	Reference name for the rule	254
Comment	Descriptive label for the rule	—
Type	It can be a definition, fact, formula, requirement, or validation	—
Expression	Presence of associated expression	—

Types of business rule in an OOM

You can define several different types of business rules.

Rule type	Describes	Example
Definition	Properties of the element in the information system	A customer is a person identified by a name and an address
Fact	Certainty, existence in the information system	A client may place one or more orders
Formula	Calculation used in the information system	The total order is the sum of all the order line costs
Requirement	Functional specification in the information system	The model is designed so that total losses do not exceed 10% of total sales
Validation	Constraint on a value in the information system	The sum of all orders for a client must not be greater than that client's allowance

Rule type	Describes	Example
Constraint	Additional check constraint on a value. Constraint business rules are used in the PDM, they are generated in the database	The start date should be inferior to the end date of a project

For more information, see section Using constraint business rules in a PDM in chapter Using Business Rules in a PDM, in the *Physical Data Model User's Guide*.

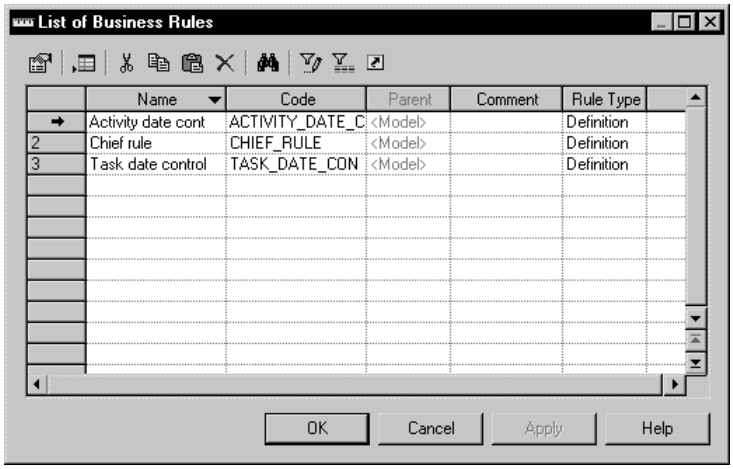
Creating a business rule in an OOM

Before you create business rules, formulate your rules by asking yourself the following questions:

- What business problems do I want to address?
- Are there any procedures that my system must respect?
- Do any specifications dictate the scope of my project?
- Do any constraints limit my options?
- How can each of these procedures, specifications, and constraints be described?
- How can each of these descriptions be classified? Possible classifications are definitions, facts, formulas, requirements or validation rules

❖ **To create a business rule in an OOM:**

1 Select Model→Business Rules to display the List of Business Rules. It displays the business rules defined for the model.



2 Click a blank line in the list.
or
Click the Add a Row tool.

An arrow appears at the beginning of the line.

3 Type a name and a code for the business rule.

4 Click Apply.

The creation of the new business rule is committed.

5 Click the new business rule line.

An arrow appears at the beginning of the line.

6 Click the Properties tool.
or
Double click the arrow at the beginning of the line.

The property sheet for the new business rule appears.

7 Click the General tab and select a business rule type from the Type dropdown listbox.

8 Click OK in each of the dialog boxes.

Applying business rules to OOM objects

From the list of business rules, you can apply a business rule to existing OOM objects. You can also apply a business rule to objects from their property sheets or lists.

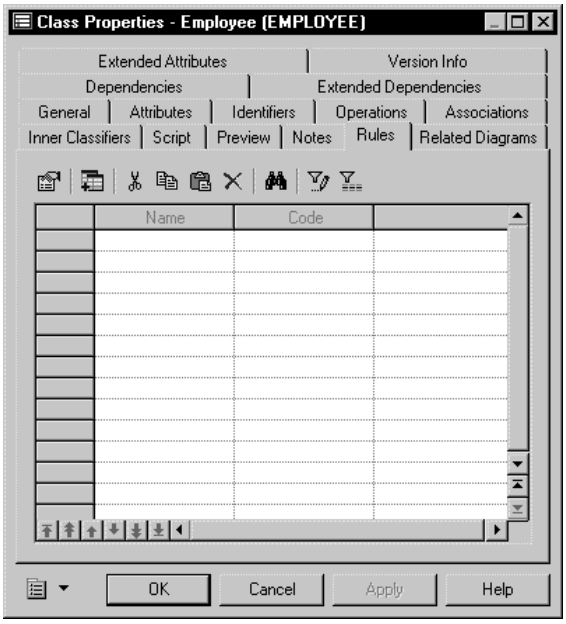
Applying a business rule to an OOM object

You can add business rules that already exist in the model, which belong to other objects.

❖ **To apply a business rule to an OOM object:**

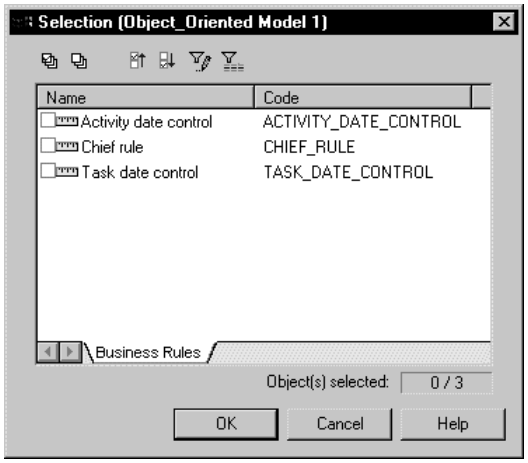
1 Double-click an object in the model to display the object property sheet.

2 Click the Rules tab to display the Rules page.



3 Click the Add Objects tool.

The Selection window appears. It contains a list of all the business rules of the model, with the exception of those that already belong to the object.



4 Select the business rules you want to add to the object.

5 Click OK.

The business rules are added to the object and appear in the list of business rules for the object.

6 Click OK.

Attaching an expression to a business rule in an OOM

A business rule typically starts out as a description. As you develop your model and analyze your business problem, you can complete a rule by adding a technical expression.

Each business rule can include two types of expression:

- Server
- Client

Expressions are used essentially in a CDM or a PDM.

For more information on expressions, see section Attaching an expression to a business rule in a PDM, in chapter Using Business Rules in the PDM, in the *Physical Data Model User's Guide*.

OOM Glossary

abstract class	A class that cannot have any direct instances
action	An action is a specification of a computable statement in a statechart diagram
action steps	Also called flow of events or basic course of actions, it refers to a text description of the sequence of actions associated with a use case
activation	Duration for processing a message
activity	Invocation of a manual or automated action
activity diagram	An activity diagram describes the flows driven by internal processing from a start point to several potential end points
actor	Outside user or related set of users that interact with a system or classifier
aggregation	A form of association that specifies a part-whole relationship between a component class and an aggregate class (example: a car has an engine and wheels)
ASP	An Active Server Page (ASP) is an HTML page that includes one or more scripts (small embedded programs) that are interpreted by a script interpreter and processed on a Microsoft Web server
association	A structural relationship that describes a set of links between objects (example: a person works for a company)
association role	The endpoint of an association, a role specifies the multiplicity and visibility between the association and the class to which it is connected
asynchronous	Request in which the sending object in a sequence diagram does not wait for a result
attribute	A named property of a class or an interface that defines the characteristics of the class or interface
attribute value	Value of an attribute (refers to an attribute of the class from which the current object is an instance)
Bean class	Class implementing the bean's business methods

Beaninfo class	Reusable software component that can visually be manipulated in a software development tool
business rule	A written statement specifying what the information system must do or how it must be structured to support business needs. It guides and documents the creation of a model (example: a government-imposed law)
class	A description of a set of objects that share the same attributes, operations, relationships, and semantics
class diagram	A class diagram is a view of a model that shows a set of packages, classes, interfaces, and their relationships that together represent the logical static design view of a system.
classifier	A classifier is a mechanism that has structural (attributes) and behavioral (operations) features. All objects that can have instances are classifiers
collaboration diagram	A collaboration diagram shows objects performing a particular goal (a functionality of the system)
component	It is an independent piece of software developed for a specific purpose but not a specific application
component diagram	A component diagram shows the organizations and dependencies among components
composition	A form of aggregation but with strong ownership and coincident lifetime of parts by the whole; the parts live and die with the whole (example: an invoice and its invoice line)
component instance	Instance of a component
constructor	An operation that creates and initializes an instance of a class
data type	A type whose values have no identity. Data types include primitive types and enumeration types
decision	Used to specify which alternate path has to be taken when several transition paths are possible
dependency	A semantic relationship between two modeling elements, in which a change to one modeling element (the influent element) may affect the semantics of the other modeling element (the dependent element)
deployment descriptor	XML file that describes the functionality of a component to an external builder tool or IDE
deployment diagram	Shows the physical configuration of run-time processing elements (nodes) on an instance level. The nodes contain instances of component that will be deployed into database, application or web servers

de-serialization class	Used to convert text or binary format into an object
diagram	A graphical view of a model or package which displays symbols
domain	Set of values for which a data item is valid. Domains are used to enforce consistent handling of data across the system
EJB	Non-visual, remotely executable components that can be deployed on a server. They provide a component architecture framework for creating distributed n-tier middleware
end	Termination point in statechart and activity diagrams
entity bean	A type of EJB server-side component. Entity beans are designed to represent data in the database; they wrap data with business object semantics, read and update data automatically
event	An event is the occurrence of something observable in the statechart diagram
exception	Signal raised in response to behavioral faults during system execution
generalization	A relationship between a more general element (the parent) and a more specific element (the child). The more specific element is fully consistent with the more general element and contains additional information.
identifier	An identifier is a class attribute, or a combination of class attributes, whose values uniquely identify each occurrence of the class
inner class	A class definition within another class definition
instance link	An instance link represents a connection between objects, it highlights the collaboration between objects
interface	A collection of operations used to specify the externally visible behavior of a class, object, or other entity. In the case of a class or object, the interface includes the signatures of the operations
J2EE	Java TM 2 Platform, Enterprise Edition (J2EE ^{TM}) is a Java platform that defines the standard for developing multi-tier enterprise applications
JavaBean	Reusable software component that can be manipulated visually in a builder tool
Java BeanInfo class	Used as a standard view of a Bean. It provides explicit information about the methods, properties, and events of a Bean
JSP	Java Server Page (JSP) is a technology for controlling the content or appearance of Web pages through the use of servlets
junction point	Exists in the statechart diagram, it is similar to the decision in the activity diagram, except that it accepts several input and output transitions

lifeline	In a sequence diagram, indicates the period during which an object exists
local interface	Extends the javax.ejb.EJBLocalObject interface. Allows beans to be tightly coupled with their clients and to be directly accessed
local home interface	Defines methods and operations used locally in a local client view. Extends the javax.ejb.EJBLocal-Home interface
message	Conveyance of information from one object to another with the expectation that activity will follow
message driven bean	Message driven beans are anonymous beans that can not be referenced by a given client, responding to JMS asynchronous messages
multiplicity	The number of allowable elements that may participate in a given relationship
node	Physical element that represents a processing resource, a real physical unit, or physical location of a deployment
node association	A structural relationship that describes a set of links between nodes
note	Symbol for displaying a comment or textual information (ex: a method body or constraint)
object	An entity with a well-defined boundary and identity that encapsulates state and behavior. Instance of a class
object diagram	The object diagram describes the structure of model elements. It is considered a diagram of instances of classes
object state	Represents an object in a particular state
object language	Contains the specifications for a particular language. It provides PowerDesigner with the syntax and guidelines for implementing stereotypes, data types, scripts and constants for an object language
object-oriented model (OOM)	Structure that is the logical design view of a software system. An OOM is a conceptual model of a software system
operation	Implementation of a service that can be requested from an object in order to affect behavior. An operation has a signature, a name, and a list of parameters (example: accelerate and brake are operations given to a car)
organization unit	Element that can represent a company, a system, a service, an organization, a user or a role. It is equivalent to the swimlane in UML
O/R mapping	Object to relational mapping. Ability to establish a correspondence between objects belonging to heterogeneous models and diagrams
package	A general purpose mechanism for organizing elements into groups

parameter	Specification of a variable that can be changed, passed, or returned. Parameters are used for operations and events
persistence	Lifetime of the instances of a class
persistent object	An object that continues to exist after the process that created it has ceased to exist
post-condition	A constraint that must be true at the completion of an operation
pre-condition	A constraint that must be true when an operation is involved
primary key class	Class providing a pointer into the database. It is linked to the Bean class and is only applicable to entity beans
procedure call	Call of a procedure to invoke an operation
realization	A semantic relationship between classifiers, in which one classifier specifies a contract that another classifier guarantees to carry out. It connects a class to an interface that supplies its behavioral specification but not its structure or implementation.
remote interface	Extends the javax.ejb.EJBObject interface. Provides the remote client view
remote home interface	Defines methods and operations used in a remote client view. Extends the javax.ejb.EJBHome interface
return	A list of values returned by a call of the operation
role	The named specific behavior of an object participating in a particular context
sequence diagram	Diagram that shows object interactions arranged in time sequence
serialization class	Used to convert an object into text or binary format
servlet	Programs that help in building applications that generate dynamic Web pages
session bean	A session bean is an EJB where each instance of a session bean is created through its home interface and is private to this client connection. The session bean instance can not be easily shared with other clients
shortcut	A shortcut represents and references an object in the same model, or across different models
signature	The name and parameters of an operation
SOAP	The Simple Object Access Protocol is a protocol based upon XML for exchange of information in a decentralized and distributed environment. Represents the invocation mechanism within a web service architecture
start	Starting point of the whole process represented in statechart and activity diagrams

state	A state represents a situation during the life of a classifier in a statechart diagram
statechart diagram	The statechart diagram is a diagram that describes a classifier behavior
stereotype	An extension of the vocabulary of the UML, it allows you to create new kinds of building blocks that are derived from existing ones but that are specific to your problem
super class	Class from which another class inherits
synchronization	Used to specify synchronization of control between two or more concurrent actions
transient object	An object that ceases to exist when the process that created it ceases to exist
transition	Route the control flow transits on to link objects in activity and statechart diagrams
UDDI	The Universal Description Discovery and Integration specification is an XML-based registry for businesses worldwide that list them on the Internet
use case	Action in a system whose task is to produce a measurable value to an actor of the system
use case diagram	Diagram that shows a set of use cases and actors and the relationships among them
visibility	Denotes how an object can be seen and used by other objects
war	Web application archive similar to a .jar file but for Web services. It is a file that compresses all the files needed for the Web application into one file
Web service	A Web service is a service offered via the Web. It is an application that accepts XML formatted requests from other systems across a network via communications protocols
WSDL	Web Services Description Language. XML formatted interface for a web service that describes what a web service is capable of, and how a client can locate and invoke that service

Index

.
.NET 406
  generate server side code 485
  generation options 486
  generation tasks 486
  Web service support 473
  WSDL extended model definition 21
.NET generation task
  compile source files 486
  Web service proxy code 486

A

abstract class 66, 542
abstract data type 682
  class 66
  multiplicity 684
  persistent 67
abstract operation 120, 542
accelerator
  preview code 26, 89, 410
access point URL
  Web service component instance 482
action 272
  activity 308
  alias 274
  check model 548
  code uniqueness 548
  condition 274
  create 275
  define 276
  duplication occurrence 549
  event arguments 273
  name uniqueness 548
  operation 273
  operation arguments 273
  properties 273
  Rose import 583
  state 265, 275
  tool 13
action (*continued*)
  transition 276
  trigger event 273
  unspecified trigger event 548
action order
  state 548
action step
  use case 39
activation 236
  attach message (sequence diagram) 250
  create 249
  create at creation of message (sequence diagram) 249
  create from diagram 249
  detach message (sequence diagram) 251
  explicit return 239
  message (sequence diagram) 236
  move 252
  overlap 251
  procedure call 250
  recursive 248
  resize 253
  return 250
  symbol 236
  tool 12
activity 306
  action 308
  change to atomic 307
  check model 550
  code uniqueness 551
  composite 307, 309
  create 310
  display preferences 315
  drag and drop into composite activity 309
  input transition missing 551
  name uniqueness 551
  organization unit 307, 308
  output transition missing 551
  properties 307
  Rose import 584, 585
  tool 14
  transition 332

activity diagram 7, 298
activity 306
convert to composite activity 301
create 301
decision 338
end 349
merge models 301, 330
object state 316
OOM 298
organization unit 321
start 303
sub-activity diagram 300
synchronization 344
transition 332
UML 298
actor
check model 526
code uniqueness 526
drag and drop in other diagram 48, 201, 229
implementation class 45, 199, 228, 508
interface 508
isolated 526
message (collaboration diagram) 527
message (sequence diagram) 527
name uniqueness 526
related diagrams 47, 200, 228
show symbol 48, 201, 229
actor (collaboration diagram) 199
create 200
display preferences 202
properties 199
tool 12
actor (sequence diagram) 227
create 229
display preferences 230
properties 228
tool 12
actor (use case diagram) 44
create 47
display preferences 49
primary actor 44
properties 45
Rose import 575, 576
secondary actor 44
tool 10
actor message
check model 527
add association class
change from association 144
add objects
tool 209
additional check
domain (OOM) 165
aggregation
association (class diagram) 140, 142
role 142
tool 11
alias
action 274
decision (activity diagram) 341
transition (activity diagram) 334
transition (statechart diagram) 280
Analysis
generate 655
object language 620
application namespace
Web service component 474
application server 437
generate O/R mapping 436
archive
Java 591
argument
event 289
message (collaboration diagram) 216
message (sequence diagram) 240
array
operation 120
parameter (class diagram) 123
parameter (statechart diagram) 286
artifact
file object (deployment diagram) 392
ASMX file
generate 486
preview 475, 480
ASP.NET 462
artifact file object 464
ASP file 462
create 464
create from class diagram 464
create from selected file object 464
default template 462
file object 463
generate 466
templatecontent 464
wizard 464
association 139
change to instance link 515
drag and drop in collaboration diagram 515
drag and drop in object diagram 515

association (*continued*)
generate PDM 676
association (class diagram) 139
add association class 144
aggregation 140, 142
change to instance link 174
composition 140, 142
container 142
create 143
create instance link (collaboration diagram) 207
create mapping 573
display preferences 150
drag and drop in object diagram 174
extended attribute 141
indicator 142
mapping 571
migrate both roles 148
migrate navigable roles 148
multiplicity 140, 141
navigability 140
ordering 140, 141
properties 139
role 139, 140
Rose import 578, 579
stereotype 139
tool 11
visibility 140, 142
association (use case diagram) 50
create 51
destination 51
display preferences 53
orientation 51
properties 51
Rose import 575, 576
source 51
tool 10
association class 139, 144
create 144
identifier 528
association role
attribute (class diagram) 108
asynchronous
message (sequence diagram) 239
atomic activity 307
atomic state 264
attach activity to organization unit (activity diagram) 327
attach activity to swimlane 327
attach message to activation 250
attribute (class diagram) 105
add 113
add getter or setter 105
add operation 111
attach 105
attach to domain 110
changeability 105, 107
check model 538, 540
check parameter 170, 539, 540
class 81
code uniqueness 538, 540
create 109, 113
data type 539, 541
default values 539, 540
derived 105
detail 110
display data type 115
display domain 115
display preferences 114
domain 105, 110, 539, 541
duplicate 113
embedded 108
enforce non divergence 29
final 539, 541
identifier 118, 535
initial value 105, 539, 541
interface 101
mapping 540, 566
migrate 91, 108, 678
migrate association role 108
migrated from 105
model options 29
multiplicity 105, 107
name uniqueness 538, 540
operation 111
parent 105
persistent 105, 108
primary identifier 105
primary key class 534
properties 105
Rose import 577, 579
static 105
update using domain 169
attribute value 181
create 183
object (object diagram) 177
properties 182
attribute visibility (class diagram) 107, 114
author
OOM 24

auto-layout
  reverse engineer 590
automatic correction
  check model 520
  OOM 522

B

bean class 410
  add internal method 415
  business method implementation 531
  check model 530, 531, 532, 533
  ejbCreate method 532
  ejbFind method 533
  ejbHome method 533
  ejbPostCreate method 532
  ejbSelect method 533
begin time
  message (collaboration diagram) 217
  message (sequence diagram) 241
bidirectional association
  C++ 626
bindingTemplate 471
BMP entity bean 409
body
  implementation operation 135
browse WSDL from UDDI 496
business method
  interface 538
business method implementation
  bean class 531
business rule (OOM) 696, 700
  apply 700
  check parameter 696
  create 698
  definition 697
  domain 165
  expression 701
  fact 697
  formula 697
  properties 697
  requirement 697
  type 697
  validation 697
businessEntity 471
businessService 471

C

C# 630
  generate 630
C# generation option
  code in .ASMX 486
  generate Web service code in .ASMX 630
C# generation task
  compile source files 630
  generate Web service proxy code 630
C++ 626
  bidirectional association 626
  generate 628
  generation 620
  unsupported features 626
call message 12
cardinality
  class 66, 70
  node association 397
  role 397
CDM
  generate 662, 668, 671
  generate package 674
  generation options 668, 670
  new 668, 671
  preserve modifications 673
  select generation objects 654
  update 668, 672
change to atomic activity 307
change to atomic state 264
changeability
  add-only 107
  attribute (class diagram) 105
  changeable 107
  frozen 107
  read-only 107
check detail
  check model 520
check model 513, 516, 518
  action 548
  activity 550
  actor 526
  actor message 527
  attribute (class diagram) 538, 540
  automatic correction 520
  bean class 530, 531, 532, 533
  check details 520
  class 527
  class generalization (class diagram) 544
  class operation 541

check model (*continued*)
component 556
component instance 558
correction 521, 522
data source (OOM) 524
decision (activity diagram) 551
domain (OOM) 523
EJB 416, 537
end 554
error 518
error list 521
event 549
file object 523
generate CDM 670
generate PDM 687
identifier 535
instance link 546
interface 536
interface generalization (class diagram) 545
interface operation 543
junction point 550
manual correction 520
message 546
node 557
object 545
object selection 517
object state 552
options 518
organization unit 553
package 525
primary key class 533, 534
primary key class attributes 534
realization 544
recheck model 520
replication 524
selection 518
start 553
state 547
synchonization 555
transition 555
use case 527
warning 518
check parameter 170
additional 171
attribute (class diagram) 539, 540
business rule (OOM) 696
properties 170
standard 170
type 170
child
generalization (class diagram) 152
generalization (component diagram) 370
generalization (use case diagram) 55
circular dependency 526
circular inheritance 525
class 65
abstract 66
abstract data type 66
actor stereotype 68
add attribute 81
add operation 81
association (class diagram) 91
association class 66
attribute 81, 147
business object 69
cardinality 66, 70
change to object (collaboration diagram) 203
change to object (object diagram) 174, 177
change to object (sequence diagram) 231, 232
change to object state 318, 512
change to state 510
check model 527
circular dependency 526
classifier 77
code uniqueness 528
component 361, 557
component diagram 361
copy and paste in collaboration diagram 511
copy and paste in object diagram 177, 511
copy and paste in sequence diagram 511
copy and paste in statechart diagram 264, 510
create 39, 45, 70, 232, 511
create an interface 100
create from an interface 71
create mapping 568, 570
display preferences 95
drag and drop in activity diagram 318, 512
drag and drop in collaboration diagram 203, 511
drag and drop in object diagram 174, 177, 511
drag and drop in sequence diagram 232, 511
drag and drop in statechart diagram 264, 510
EJB 423
enumeration stereotype 68
exception stereotype 68
final 66, 544
generate 66
generate PDM 676
identifier 116

class (*continued*)
implement interface 133
implementation 506, 508
implementationClass stereotype 68
inherited association 91
inner classifier 77
inner to 66
instantiation 505, 511
JavaBean 69
mapping 535, 565, 566
metaclass stereotype 68
migrate attribute 91
name uniqueness 528
object (sequence diagram) 232
object state 318
operation 82, 126
paste as shortcut in object diagram 177
persistent 66, 67, 528
powertype stereotype 68
preview code 89
process stereotype 68
properties 66
realization 71, 161
related diagrams 70
Rose import 576, 577, 578
signal stereotype 68
state 510
stereotype 68
storage 69
thread stereotype 68
tool 11
type 66, 69
type stereotype 68
utility 69
utility stereotype 68
visibility 66, 69, 528
visual object 69
class diagram 4, 64
association 139
attribute 105
class 65
component 357
create 64
create a JSP 455
create a servlet 446
create an ASP.NET 464
create an EJB 412
create component 365
create for component 364
create Web service 476
class diagram (*continued*)
dependency 156
domain 165
generalization 151
identifier 116
interface 96
OOM 64
open 364
operation 120
parameter 122
realization 161
servlet 446
UML 64
Web service 480
classifier 77
attach to data type 83, 84, 88
attach to return type 83, 86
attribute data type 84
class 77
empty 528
fully qualified name 83
interface 77
operation return type 86
parameter data type 88
state 264
use case 77
CLASSPATH
J2EE_HOME 408
JAVA_HOME 408
CMP entity bean 409, 427
CocoBase
extended model definition 436
code
body 594
comment 592
Java 594
persistent 67, 108
preview 26, 89, 410
code editor
java source code 604
code generation
EJB 416, 419
code uniqueness
action 548
activity 551
actor 526
attribute (class diagram) 538, 540
class 528
component 557
component instance 558

code uniqueness (*continued*)
decision (activity diagram) 552
end 554
event 549
identifier 535
interface 536
junction point 550
message 546
node 558
object 545
object state 552
operation 541, 543
organization unit 553
start 554
state 547
synchronization 555
use case 527
collaboration diagram 5
actor 199
convert to sequence diagram 197
create 197
instance link 207
message 213
multiple 203
object 203
OOM 196
UML 196
comment
generate 633
Java code 592
Javadoc 600
Web service component instance 483
compile
reverse engineering Java 596
complex data type 664, 680
.NET 479
Java 479
component 356
change to component instance 388, 511
change to state 510
check model 556
class 361, 557
class diagram 357, 364
code uniqueness 557
component instance 511, 559
copy and paste in deployment diagram 511
copy and paste in statechart diagram 264, 510
create 365, 511
create class diagram 364
create from class diagram 365
component (*continued*)
create in component diagram 366
deploy to node 367
display preferences 369
document stereotype 357
drag and drop in deployment diagram 388, 511
drag and drop in statechart diagram 264, 510
executable stereotype 357
file objects 362
file stereotype 357
instantiation 511
interface 359, 538, 557
isolated 557
library stereotype 357
name uniqueness 557
open class diagram 364
properties 357
related diagrams 359
Rose import 586, 587
state 510
stereotype 357
table stereotype 357
tool 14
transaction 357
type 357, 358
update class diagram 364
view operations 363
Web service 357
component diagram 7, 354
ASP.NET 462
class 361
component 356
create 355
dependency 374
EJB 408
file object 362
generalization 370
interface 359
J2EE 406
JSP 454
OOM 354
servlet 445
UML 354
Web service 473
component instance 388
cardinality 389
check model 558
code uniqueness 558
component 559

component instance (*continued*)
component instantiation 511
create 389
display preferences 391
duplicated in a node 559
instance of component 511
isolated 559
name uniqueness 558
properties 389
tool 15
Web service 389, 482
component interface
operation 363
component type
import WSDL 493
composite activity 301, 307, 309
close 314
create 312
go up one level 314
open 314
package 32
start 551
composite state 259, 264, 266
close 270
create 268
go up one level 270
open 270
package 32
start 548
composition
association (class diagram) 140, 142
role 142
tool 11
conceptual data type 663
condition
action 274
conditional branch 339, 341
decision (activity diagram) 341
message (collaboration diagram) 217
message (sequence diagram) 241
missing on transition 556
transition (activity diagram) 334, 339
transition (statechart diagram) 280
useless on transition (activity diagram) 556
conditional branch 339
condition 339, 341
factorize condition 339
configure
CDM options 669
PDM options 686
constraint
business rule (OOM) 696
constructor
add 128, 130
default 128
interface 536
modifier 529
operation 128, 130
return type 529
container
association (class diagram) 142
control flow
message (collaboration diagram) 213
message (sequence diagram) 236, 239
convert to composite 259, 301
copy
object language 22
copy constructor 130
correction
automatic 522
check model 521, 522
manual 521
OOM 520, 522
Create
message (collaboration diagram) 216
message (sequence diagram) 238, 243
create a new message
tool 209
create associations
reverse engineering Java 592
reverse engineering PowerBuilder 607
create in class diagram
reverse engineering WSDL 494
create in component diagram
reverse engineering WSDL 494
create in deployment diagram
reverse engineering WSDL 494
create internal link 149
create symbol for association (class diagram) 149
create symbols
reverse engineering Java 592
reverse engineering PowerBuilder 607
reverse XML 614
creation tool 10
Criteria
mapping 566, 570

D

data source (OOM)
  check model 524
  create 562
  define database connection 562
  define models 562
  differing DBMS 525
  existence of PDM 525
  models 561
  name uniqueness 525
  O/R mapping 560, 561
  ODBC 561
  properties 561
data type
  attribute (class diagram) 84, 105, 539, 541
  conceptual 663
  default 28
  domain (OOM) 165, 167
  model options 29
  object language 663
  options 28
  parameter (class diagram) 88, 123
  parameter (statechart diagram) 286
  persistent 108
  select 167
  translate 663, 676
  Web service 478
data type generation
  multiplicity 666, 682
database connection 562
decision (activity diagram) 338
  alias 341
  check model 551
  code uniqueness 552
  condition 341
  conditional branch 338
  create 341
  display preferences 343
  incomplete on transition 552
  merge 338, 339
  name uniqueness 552
  properties 340
  Rose import 586
  tool 14
  type 340
default attribute value 539, 540
default classifier
  statechart diagram 258
default constructor 128
default data type 28
default destructor 128
default diagram 9
  OOM 24
  package 32
default template
  ASP.NET component 462
  JSP component 454
default value
  parameter (class diagram) 123
  parameter (statechart diagram) 286
deferred event 272
  state 265
delete rule
  PDM generation option 688
dependency (class diagram) 156
  create 158
  display preferences 160
  properties 157
  Rose import 579
  stereotype 157
  tool 11
dependency (component diagram) 374
  create 375
  display preferences 377
  properties 374
  tool 14
dependency (deployment diagram) 401
  create 402
  display preferences 404
  properties 401
  tool 15
dependency (object diagram) 190
  create 190
  display preferences 193
  properties 190
  tool 12
dependency (use case diagram) 59
  create 60
  display preferences 62
  extend 60
  include 60
  properties 59
  Rose import 576
  stereotype 60
  tool 10
dependency stereotype (class diagram)
  ejb-ref 158
  Java 158
  sameFile 158

dependency stereotype (component diagram)
ejb-ref 375
Java 375
sameFile 375
deploy
.NET Web services 486
deploy component
create component instance 367
create shortcut of component 367
deploy component to node 367
deployment descriptor
EJB 422
JAR 422
reverse engineering Java 592
Web service 485
XML file 422
deployment diagram 8, 380
component instance 388
create 381
dependency 401
file object 392
node 382
node association 396
node diagram 382
OOM 380
Web service 482
derived
class 566, 679
de-serialization class 479
Web service class 480
destination of association (use case diagram) 51
destination of transition (activity diagram) 333
destination of transition (statechart diagram) 279
Destroy
message (collaboration diagram) 216
message (sequence diagram) 239, 244
destructor
operation 128
detach activity to organization unit (activity diagram) 327
detach activity to swimlane 327
detach message from activation 251
detach OOM from workspace 24
detail
attribute (class diagram) 110
diagram
model 9
package 10, 34
diagram type
package 34
direction
parameter (class diagram) 123
parameter (statechart diagram) 287
directory
reverse engineering Java 591, 597
disable swimlane mode 321
display
error message 520
warning message 520
display preference
activity 315
actor (collaboration diagram) 202
actor (sequence diagram) 230
actor (use case diagram) 49
association (class diagram) 150
association (use case diagram) 53
attribute (class diagram) 114
class 95
component 369
component instance 391
decision (activity diagram) 343
dependency (class diagram) 160
dependency (component diagram) 377
dependency (deployment diagram) 404
dependency (object diagram) 193
dependency (use case diagram) 62
end (activity diagram) 352
end (statechart diagram) 296
generalization (class diagram) 155
generalization (component diagram) 373
generalization (use case diagram) 58
instance link (collaboration diagram) 212
instance link (object diagram) 189
interface 104
junction point 293
message (collaboration diagram) 222
message(sequence diagram) 254
node 387
node association 400
object (collaboration diagram) 206
object (object diagram) 180
object (sequence diagram) 235
object state 320
operation 138
organization unit 331
package 34
realization 164
start (activity diagram) 305
start (statechart diagram) 263
state 271

display preference (*continued*)
swimlane 331
transition (activity diagram) 337
transition (statechart diagram) 284
use case 43
diverge from domain (OOM) 29
do event 272, 285
documentation
Rose import 575
domain (OOM) 165
additional checks 165
attribute (class diagram) 105, 110, 539, 541
business rules 165
check model 523
check parameter 170
create 166
data type 165, 167
diverge from 29
enforce non divergence 29
model options 29
multiplicity 165, 166
persistent 165, 166
properties 165
standard checks 165
update attributes 169
duplicate operation 127
duplicated occurrence
action 549
transition 556

E

EAServer
extended model definition 436
EJB 408
attached classifiers 557
bean class 410
check model 416, 537
cmp-fields 419
cmr-fields 419
code generation 416, 419
create 412
create from a class diagram 412
create from selected class 412
create operation from bean class 427
create operation from interface 427
define class 423
define interface 423
define operation 426
EJB (*continued*)
deployment descriptor 422
generate 429
generate deployment descriptor 435
generate JAR 438
generate O/R mapping 436
generate source code 433
initialization 416, 417
linked method 419
local home interface 410
local interface 410
operation stereotype 427
persistence 431
preview 410
primary key class 410
properties 410
remote home interface 410
remote interface 410
reverse engineering 439, 442
reverse options 439
source code 431
stereotype 423
synchronization 416, 418
transaction 412
version 408
view operation 427
wizard 412
EJB deployment descriptor
element 440
EJB entity bean 409
EJB message driven bean 409, 410
EJB session bean 409
stateful 409
stateless 409
ejbCreate method 427
bean class 532
home interface 531
interface 537
ejbFind method
bean class 533
interface 537
ejbFinder method 427
ejbHome method
bean class 533
ejb-jar.XML 438
ejbPostCreate method 427
bean class 532
ejb-ref
dependency stereotype (class diagram) 158

ejb-ref (*continued*)
dependency stereotype (component diagram) 375
ejbSelect method 427
bean class 533
element
EJB deployment descriptor 440
embedded
class generation 665, 681
data type generation 664, 665, 680, 681
embedded attribute 108
enable swimlane mode 321
encoding style
Web service component 475
end (activity diagram) 349
check model 554
code uniqueness 554
create 350
display preferences 352
missing transition 554
name uniqueness 554
properties 349
Rose import 585
tool 14
end (statechart diagram) 294
check model 554
code uniqueness 554
create 294
display preferences 296
missing transition 554
name uniqueness 554
properties 294
Rose import 583
tool 13
end time
message (collaboration diagram) 217
message (sequence diagram) 241
enforce non divergence 29
entity/relationships
notation 670
OOM 662
entry event 272, 285
error
OOM 516
severity 516
error list
check model 521
navigate 521
error message
display 520
error message (*continued*)
OOM 522
event 285
arguments 289
check model 549
code uniqueness 549
create 289
name uniqueness 549
parameter 285
properties 286
trigger event 285
unused 550
event argument
action 273
exception
implementation operation 135
transition (activity diagram) 334
transition (statechart diagram) 280
use case 39
exit event 272, 285
explicit return
activation 239
expression
business rule (OOM) 701
extend
dependency (use case diagram) 60
final class 544
extended attribute
association (class diagram) 141
JMS 475
UDDI 475
extended dependencies 16
extended model definition 21, 623
CocoBase 436
EAServer 436
WebLogic 436
extension point
use case 39

F

file encoding
Java reverse engineering 591
file object
ASP.NET 463
ASP.NET stereotype 463
check model 523
component 362
component diagram 362

file object (*continued*)
JAX-RPC 488
JSP 455
location 524
name uniqueness 524
file object (deployment diagram) 392
artifact 392
create 393
generate 392
Rose import 587
filename
OOM 24
final
attribute (class diagram) 539, 541
class 544
find in Browser 266, 309
first diagram
create OOM 18
FK index name
PDM generation option 689
FK threshold
PDM generation options 689
flow type
transition (activity diagram) 333, 334
transition (statechart diagram) 279, 280
foreign key 688
generate 690
index 690
name 690
name template for generation 688
fork 344
input flow 344
output flow 344
format
variable 94
fully qualified name
classifier 83

G

generalization (class diagram) 151, 679
check model 544, 545
child 152
class 544
create 152
display preferences 155
extend final class 544
interface 545
Java 544
generalization (class diagram) (*continued*)
parent 152
PowerBuilder 544
properties 152
redundancy 544, 545
Rose import 578, 579
stereotype 152
tool 11
visibility 152
generalization (component diagram) 370
child 370
create 371
display preferences 373
parent 370
properties 370
stereotype 370
tool 14
visibility 370
generalization (use case diagram) 54
child 55
create 56
display preferences 58
parent 55
properties 55
tool 10
visibility 55
generate
.NET commands 486
.NET options 486
.NET Web services 486
Analysis 655
ASMX file 485
ASP.NET 466
C# 620, 630
C# options 630
C# tasks 630
C++ 620, 628
check model 622
class 66
configure CDM options 669
configure PDM options 686
EJB 429, 433
EJB deployment descriptor 435
external shortcut 652
file object (deployment diagram) 392
foreign key 690
foreign key name 688
implementation OOM 655
implementation WSDL 489
index parameter 689

generate (*continued*)
interface 97
interface WSDL 489
Java 620, 638
Java BeanInfo 72
Java commands 489
Java options 488, 635
Java tasks 637
JavaBean 72
Javadoc 633
Javadoc comments 633
JSP 457
JSP Web deployment descriptor 457
language 622
model 621
new CDM 668, 671
new model 652
new OOM 655, 656
new PDM 685, 691
object language 620
OOM 652, 657
OOM from OOM 657
OOM to OOM 655
package 621
PowerBuilder 620, 641
PowerBuilder options 640
primary key 690
reference parameter 688
select object 621
servlet 448
servlet Web deployment descriptor 451
servlet Web service 491
SRU file 641
standard Web service 491
stateless session bean Web service 491
table parameter 688
target models 652
update CDM 669, 672
update existing OOM 655
update model 652
update PDM 685, 692
VB.NET 620, 643
VB.NET options 643
VB.NET tasks 643
Web services 485, 490
WSDL 620
XML 620, 647
XML-Data 647
XML-Data options 647
XML-DTD 646
generate (*continued*)
XML-Schema 646
generate CDM 662, 668, 669
options 670
package 674
preserve modifications 673
select objects 654
generate JAR 438
generate O/R mapping 436
generate PDM 675, 684, 686
options 687
package 694
preserve modifications 693
generate server side code
.NET 485
Java 488
generation
persistent 108
generation target 623
generation task 622
generation template language 620
Getter
operation 111
group symbols 225, 324

H

hierarchy
package 32
home interface
method implementation 531
horizontal
swimlane 331

I

icon
activity diagram 9
attribute visibility (class diagram) 114
class diagram 9
collaboration diagram 9
component diagram 9
deployment diagram 9
object diagram 9
operation 138
sequence diagram 9
statechart diagram 9
use case diagram 9

identifier 116
add attributes 118
association class 528
attribute (class diagram) 535
check model 535
class 116
code uniqueness 535
create 118
inclusion 536
name uniqueness 535
primary identifier 116
properties 116
ignore comments
reverse engineering Java 592
reverse engineering PowerBuilder 607
ignore operation body
reverse engineering Java 592
reverse engineering PowerBuilder 607
implementation 655
interface 506, 508
stereotype 152, 370
Web service 476
Web service component 474
implementation class
actor 45, 199, 228, 508
use case 39, 506
implementation code 135
implementation operation 135
body 135
create 133
exception 135
exception classifier 135
post-condition 135
pre-condition 135
specification 135
implementation WSDL
generate in Java 489
implicit
stereotype 139
import
model 574
Rose 574
import extended model definition 437
import interface WSDL
Web service component instance 483
import WSDL 493
include
dependency (use case diagram) 60
include composite activities
tool 309
include composite states
tool 266
inclusion
identifier 536
incomplete junction point 550
index
foreign key 690
generate 689
primary key 690
indicator
association (class diagram) 142
inheritance 105, 127, 679
circular 525
multiple 544
inherited attribute (class diagram) 105, 566, 679
inherited operation 120
inialization
EJB 417
servlet 448
initial value
attribute (class diagram) 105
inner classifier 77
attach 79
detach 80
inner link 77
reverse engineering Java 590
reverse engineering XML 590
inner link 77
inner classifier 77
tool 11
inner to
class 66
interface 97
input flow
fork 344
join 344
input transition
missing on activity 551
state 547
instance link
check model 546
instance of association 515
redundant 546
instance link (collaboration diagram) 207
create 207, 210
display preferences 212
drag and drop association (class diagram) 207
message 209, 547
parallel instance links 208
properties 209

instance link (collaboration diagram) (*continued*)
Rose import 580, 581
sequence number 207
show symbols 208
tool 12
instance link (object diagram) 185
create 187
display preferences 189
properties 186
tool 12
instantiation
class 511
component 511
interface 96
actor 508
add attribute 101
add operation 101
attach inner classifier 101
attribute (class diagram) 101, 147
business method 538
check model 536
classifier 77
code uniqueness 536
component 359, 538, 557
component diagram 359
constructor 536
create 39, 45, 99
create a class 71
create from a class 100
create method 537
display preferences 104
EJB 423
finder method 537
generate 97
implementation 506, 508
inner classifier 77
inner to 97
metaclass stereotype 98
name uniqueness 536
navigability 536
operation 103
powertype stereotype 98
preview code 89
process stereotype 98
properties 97
realization 100, 161
related diagrams 98
Rose import 577, 578
stereotype 98
thread stereotype 98
interface (*continued*)
tool 11
use case 506
utility stereotype 98
visibility 97, 98
Web service 476
Web service component 474
interface WSDL
generate in Java 489
preview 480
internal transition 264, 272
isolated actor 526
isolated component instance 559
isolated object 545

J

J2EE 406
J2EE application server 408
J2EE HOME
path 408
JAR 438
reverse engineering Java 598
Java 633
code 594
code comment 592
EJB 2.0 633
EJB subfamily 633
generate 638
generate server side code 488
J2EE 406
JSP 454
reverse engineering 591
reverse file 595
script 93
servlet 445
Web service support 473
WSDL extended model definition 21
zip 598
Java BeanInfo 72
create 74
generate 72
JavaBean 530
Java generation option 488, 635
add Java classes code 635
generate CMP field 635
generate CMR field 635
generate WSDL and proxy 488
Jar Web component classes 635

Java generation option (*continued*)
primary sort 635
type sort 635
visibility sort 635
XRPCC input 488
Java generation task
compile Java sources 637
create EAR 637
create EJB jar 637
create WAR 637
generate Javadoc 637
generate proxy class using XRPCC 489
generate Web service proxy code 637
generate WSDL 637
generate WSDL using XRPCC 489
run J2EE verifier 637
run Java 637
JAVA HOME
path 408
Java IDE 408
Java reverse engineering
compile 596
directory 597
file encoding 591
options 592
JavaBean
BeanInfo 530
class 69
generate 72
Javadoc 633
comment 600
comments 633
JAXM 471, 476, 488
JAX-RPC 471, 476
JDK 408
library 599
load 599
model 599
open 599
JMS 410
extended attributes 475
join 344
input flow 344
output flow 344
JSP 454
create 455
create from a class diagram 455
create from selected file object 455
file object 455
generate 457
JSP (*continued*)
generate deployment descriptor 457
reverse engineering 460
templatecontent 455
wizard 455
JSP component
default template 454
JSP file object 454
JSP file object
artifact 455
JSR specification 471, 476, 485, 488
junction point 291
check model 550
code uniqueness 550
create 291
display preferences 293
incomplete 550
name uniqueness 550
properties 291
tool 13
justify
variable 94

K

key index name
PDM generation options 689
keyword
attribute visibility (class diagram) 114

L

language event
operation 120
library
JDK 599
PowerBuilder 608
reverse engineering Java 592
reverse engineering PowerBuilder 607
linked method 416, 419
list of value
standard checks 182
LISTVAL
check parameter 171
local home interface 410
local interface 410
Local2SOAP 478
Local2XSD 478

location
file object 524

M

manual correction
check model 520
OOM 521
mapping
association (class diagram) 571
attribute (class diagram) 540, 566
class 535, 565
class source 566
create for a class 568, 570
create for an association (class diagram) 573
Criteria page 566, 570
object to relational 560
operation 570
query 570
SQL queries 567, 572
mark classifiers
reverse engineering Java 592
reverse engineering PowerBuilder 607
marker
attribute visibility (class diagram) 114
operation visibility 138
merge model
activity diagram 330
sequence diagram 225
transition (statechart diagram) 278
merise
notation 670
OOM 662
message
code uniqueness 546
name uniqueness 546
message (collaboration diagram) 213
actor 527
arguments 216
begin time 217
check model 546
condition 217
control flow 213
create 217
Create 216
create sequence number 219
decrease sequence number 222
Destroy 216
display preferences 222
message (collaboration diagram) (*continued*)
end time 217
increase sequence number 221
instance link 547
operation 216, 514
properties 215
recursive 213
reverse direction 215
Self-Destroy 216
sequence number 215, 546
support delay 217
tool 12
message (sequence diagram) 236
activation 236
actor 527
arguments 240
asynchronous 239
begin time 241
check model 546
condition 241
control flow 236, 239
create 242
Create 238, 243
create object 243
Destroy 239, 244
display preferences 254
end time 241
operation 240, 514
procedure call 239
properties 238
recursive 246, 247, 248
return 239, 242
reverse direction 238
Self-Destroy 239, 244
sequence number 238, 546
support delay 30, 241
tool 12
undefined 239, 242
message driven bean
Web service 476
message frequency (sequence diagram)
Rose import 582
message synchronization (collaboration diagram)
Rose import 580
message synchronization (sequence diagram)
Rose import 581
migrate
association role (class diagram) 147
migrate both roles
association (class diagram) 148

migrate navigable roles
association (class diagram) 148
migrated attribute 147
migrated from
attribute (class diagram) 105, 108
migration 678
MINMAX
check parameter 171
model
CDM options 669
check 518
copy extended model definition 21
copy object language 17
create 17
diagram 9, 10
import 574
JDK 599
open 23
options 28
package 9, 10
PDM options 686
PowerBuilder 608
preview code 26
properties 24
share extended model definition 21
share object language 17
model option
attribute (class diagram) 29
check 29
class 28
data type 28, 29
domain (OOM) 29
message (sequence diagram) 30
name/code case sensitive 28
preview editable 28
rules 29
use data type full name 29
modifier
constructor 529
operation 543
move down
generation tasks 622
move up
generation tasks 622
multiple
collaboration diagram 203
object diagram 176
multiple inheritance 544
multiplicity
abstract data type 684
association (class diagram) 140, 141
attribute (class diagram) 105, 107
data type generation 666, 682
domain (OOM) 165, 166
node association 397
role 141, 397

N

name uniqueness
action 548
activity 551
actor 526
attribute (class diagram) 538, 540
class 528
component 557
component instance 558
data source (OOM) 525
decision (activity diagram) 552
end 554
event 549
file object 524
identifier 535
interface 536
junction point 550
message 546
node 558
object 545
object state 552
operation 541, 543
organization unit 553
role 529
start 554
state 547
synchronization 555
use case 527
name/code case sensitive 28
check model 28
namespace
package 32
Web service 476
naming convention 425
navigability
association (class diagram) 140
interface 536
network address
node 484

new version
object language 24
node 382
check model 557
code uniqueness 558
component deployed 367
component instance 384, 558
create 384
display preferences 387
empty 558
name uniqueness 558
network address 484
node diagram 382
properties 384
Rose import 587
tool 15
node association 396
cardinality 397
create 397
display preferences 400
multiplicity 397
properties 396
role 396
Rose import 587
stereotype 396
tool 15
node diagram 382
close 383
create 383
deployment diagram 382
go up one level 383
open 383
notation
entity/relationships 670
merise 670

O

O/R mapping 356, 560
association (class diagram) 564, 571
attribute (class diagram) 564
class 564, 565
column 564
data source (OOM) 561, 562
operation 564, 570
table 564
object
change to object state 318
check model 545
object (*continued*)
class instantiation 511
code uniqueness 545
create 318
drag and drop in activity diagram 318, 513
generate 621
instance of class 511
isolated 545
name and code 177
name uniqueness 545
object state 317, 318
undefined on object state 553
object (collaboration diagram) 203
create 204
display preferences 206
properties 204
Rose import 580, 581
tool 12
object (object diagram) 176
attribute value 177
create 178
display preferences 180
properties 177
tool 12
object (sequence diagram) 231
class instantiation 232
code 232
create 233
display preferences 235
name 232
properties 232
Rose import 582
tool 12
object diagram 5, 174
attribute value 181
create 175
dependency 190
instance link 185
multiple 176
object 176
OOM 174
UML 174
object language
C# 630
C++ 626
change 22
compare 16
copy 22
create 16
data type 663

object language (*continued*)
generate 620
Java 633
merge 16
new version 24
OOM 24
PowerBuilder 640
replace 24
replace from previous version 24
reverse engineering 590
share 22
VB.NET 643
object lifeline 231
object persistence 663, 677
complex data type 664, 678, 680
simple data type 664, 678
object selection
check model 517
object state 316
check model 513, 552
class 318
code uniqueness 552
create 319, 512
display preferences 320
name uniqueness 552
object 317, 318
properties 317
Rose import 585
state of an object 513
tool 14
transition (activity diagram) 332
undefined object 553
OMG
UML xi, 3
OOM 4
activity diagram 298
all diagrams 8
author 24
check model 516
check model options 516
class diagram 64
collaboration diagram 196
component diagram 354
correction 520, 522
create 17
default diagram 24
deployment diagram 380
detach from workspace 24
error 516, 522
extended model definition 21
OOM (*continued*)
filename 24
first diagram 18
generate 652, 655
generate CDM 662
generate from OOM 657
generate PDM 675
generation option 656, 657
new CDM 671
new PDM 691
object diagram 174
object language 24
objects 10
open 23
options 28
overview 2
PDM objects 675
properties 24
sequence diagram 224
statechart diagram 256
tool palette 10
translate to PDM 675
update 659
update CDM 672
update PDM 692
use case diagram 36
validate 516
version 24
warning 516, 522
operation 120
abstract 120, 542
activate 131
add 136
add from parent class 127
array 120
attribute (class diagram) 111
check model 541, 543
class 82, 126, 529, 541
code uniqueness 541, 543
component 363
component interface 363
constructor 128, 130
constructor stereotype 121
copy constructor 130
create 125, 216, 240, 514
create stereotype 121
deactivate 131
default constructor 128
default destructor 128
destroy stereotype 121

operation (*continued*)
display preferences 138
duplicate 127, 136
EJB 426
ejbCreateMethod 121
ejbFinderMethod 121
ejbSelectMethod 121
final 120
Getter 111
icon 138
implementation 135, 529
interface 103, 529, 543
language event 120
mapping 570
message (collaboration diagram) 216, 514
message (sequence diagram) 240, 514
modifier 543
name uniqueness 541, 543
override 127
overriding 543
parameter data type 542, 543
parent 120
properties 120
read-only 120
related diagrams 122
return type 86, 120, 542, 543
Rose import 577, 579
Setter 111
signature 542
static 120
static initializer 132
stereotype 121
storedFunction 121
storedProcedure 121
visibility 120, 122, 138
Web service method 120
operation return type
classifier 86
operation signature
overloading 542
operation visibility
keyword 138
marker 138
options
configure for generated CDM 669
configure for generated PDM 686
data type 28
generate 622
generate CDM 670
generate PDM 687
options (*continued*)
model 28
OOM 28
PowerBuilder 607
reverse engineering EJB 439
reverse engineering Java 592
reverse engineering JSP 460
reverse engineering servlet 452
reverse engineering WSDL 494
reverse engineering XML 614
ordering
association (class diagram) 140, 141
role 141
organization unit (activity diagram) 321
activity 307, 308
attach activity 327
check model 553
circular dependency 553
code uniqueness 553
create 323
detach activity 327
display preferences 331
name uniqueness 553
open property sheet 322
parent organization 322
properties 322
stereotype 323
swimlane 321, 322, 328
tool 14
orientation
association (use case diagram) 51
output flow
fork 344
join 344
output transition
missing on activity 551
overloading operation 542
override 127
overriding operation 543
overview
OOM 2

P

package 32
check model 525
circular inheritance 525
class visibility 69
default diagram 32, 34

package (*continued*)
display preferences 34
generate CDM 674
generate PDM 694
hierarchy 32
import WSDL 493
interface visibility 98
model 9, 10
namespace 32
operation visibility 122
preview code 26, 33
properties 32
Rose import 575, 581, 584, 588
shortcut code uniqueness 526
stereotype 32
sub-package 32
tool 10
parallel instance link (object diagram) 185
parallel transition (activity diagram) 332
parameter (class diagram) 122
array 123
create 124
data type 88, 123
default value 123
direction 123
parameter type 123
parent 123
properties 123
WSDL data type 123
parameter (statechart diagram) 286
array 286
create 287
data type 286
default value 286
direction 287
event 285
parameter type 286
parent 286
properties 286
parameter direction (class diagram)
In 123
In/Out 123
Out 123
parameter direction (statechart diagram)
In 287
In/Out 287
Out 287
parent
attribute (class diagram) 105
circular dependency 553
parent (*continued*)
generalization (class diagram) 152
generalization (component diagram) 370
generalization (use case diagram) 55
operation 120
parent organization
organization unit 322
PBL
generate 640
PDM
data type 676
generate 675, 684, 691
generate from association 676
generate from class 676
generate package 694
generation options 685, 687
new 691
preserve modifications 693
update 692
persistence
attribute migration 678
OOM to CDM 663
OOM to PDM 677
persistent
abstract data type 67
attribute (class diagram) 108, 664, 680
class 67, 528
class generation 108, 665, 680, 682
code 67, 108
data type 108
domain (OOM) 165, 166
length 108
precision 108
PK index name
PDM generation options 689
post-condition
implementation operation 135
use case 39
PowerBuilder 128, 131, 640
generate 641
library 608
load 608
model 608
objects 607
open 608
options 607
reverse engineering 607
SRU 641

PowerBuilder generation 620
options 640
PBL application 640
select object 641
SRU file 640
pre-condition
implementation operation 135
use case 39
preserve modifications 659
generate CDM 673
generate PDM 693
preview
accelerator 26, 89, 410
ASMX file 475, 480
bookmark 26, 89, 410
code 26, 89, 410
EJB 410
interface WSDL 480
package 33
refresh 26, 89, 410
reverse engineering Java 27
select generation target 89, 410
show generation options 26, 89, 410
Web service 475
Web service class 480
preview editable 28
preview WSDL
import WSDL 493
previous version model 77
primary identifier 116
attribute (class diagram) 105
create 117
primary key
generate 690
index 690
name 690
primary key class 410, 534
attribute (class diagram) 534
check model 533, 534
definition 533
private
class visibility 69
interface visibility 98
operation visibility 122
procedure call
activation 250
message (sequence diagram) 239
recursive 248
protected
class visibility 69
interface visibility 98
operation visibility 122
public
class visibility 69
interface visibility 98
operation visibility 122

Q

query 570

R

Rational Rose
open model 574
version 574
realization 161
check model 544
class 71, 161
create 162
display preferences 164
interface 100, 161
properties 161
redundancy 544
stereotype 161
tool 11
rebuild data type links 149
receiver 236
recheck model
check model 520
recursive
activation 248
procedure call 248
recursive message (collaboration diagram) 213
recursive message (sequence diagram)
create 246
return 247
undefined 247
redundancy
generalization (class diagram) 544, 545
realization 544
redundant instance link 546
reference
foreign key name 688
generate 688
reflexive instance link (object diagram) 185

reflexive transition (activity diagram) 332
reflexive transition (statechart diagram) 278
related diagram 502
  actor 47, 200, 228
  add 503
  class 70
  component 359
  interface 98
  oom objects 502
  open 504
  operation 122
  use case 41
remote home interface 410
  method 538
remote interface 410
replace object language 24
replication
  check model 524
  partial replication 524
return
  activation 250
return message (sequence diagram) 12, 239, 242
return recursive message (sequence diagram) 247
return type
  constructor 529
  operation 86, 120, 542, 543
reuse a message
  tool 209
reverse direction
  message (collaboration diagram) 215
  message (sequence diagram) 238
reverse engineering 590
  .class 591
  .jar 591
  .java 591
  .zip 591
  auto-layout 590
  code 594
  EJB 439, 442
  EJB deployment descriptor 440
  Java 591
  Javadoc comment 600
  JSP 460
  JSP deployment descriptor 460
  object language 590
  objects from PowerBuilder 609
  objects from SRU files 611
  PBL 609
  servlet 452
  servlet deployment descriptor 452
reverse engineering (*continued*)
  SRU 611
  synonym creation 590
  Web service 493
reverse engineering Java 595
  class 591
  code preview 27
  compile 596
  dependency 591
  deployment descriptor 592
  directory 597
  JAR 598
  options 592
  preview 27
  without body code 594
  zip 598
reverse engineering PowerBuilder 607
reverse engineering XML 613, 615
role
  aggregation 142
  association (class diagram) 139, 140
  cardinality 397
  composition 142
  migrate from association (class diagram) 147
  multiplicity 141, 397
  name uniqueness 529
  node association 396
  ordering 141
Rose import 575
  action 583
  activity 584, 585
  actor (use case diagram) 575, 576
  association (class diagram) 578, 579
  association (use case diagram) 575, 576
  asynchronous message (collaboration diagram) 580
  asynchronous message (sequence diagram) 581
  attribute (class diagram) 577, 579
  balking message (collaboration diagram) 580
  balking message (sequence diagram) 581
  class 576, 577, 578
  component 586, 587
  decision (activity diagram) 586
  dependency (class diagram) 579
  dependency (use case diagram) 576
  destruction marker 581
  documentation 575
  end (activity diagram) 585
  end (statechart diagram) 583

Rose import (*continued*)
file object (deployment diagram) 587
generalization (class diagram) 578, 579
instance link (collaboration diagram) 580, 581
interface 577, 578
message frequency (sequence diagram) 582
message synchronization (collaboration diagram) 580
message synchronization (sequence diagram) 581
node 587
node association 587
object (collaboration diagram) 580, 581
object (sequence diagram) 582
object state 585
operation 577, 579
package 575, 581, 584, 588
procedure call 581
return message 581
self-destroy message 581
simple message (collaboration diagram) 580
simple message (sequence diagram) 581
start (activity diagram) 585
start (statechart diagram) 583
state 583
statechart diagram 582
stereotype 575
synchronous message (collaboration diagram) 580
synchronous message (sequence diagram) 582
time out message (collaboration diagram) 580
time out message (sequence diagram) 581
transition (activity diagram) 585
transition (statechart diagram) 583, 584
use case 576
zoom 575
rule
constraint 696
RULES
check parameter 171

S

sameFile
dependency stereotype (class diagram) 158
dependency stereotype (component diagram) 375
save generation dependencies
generate CDM 670
save generation dependencies (*continued*)
PDM generation option 687
script
class 93
customize 93
interface 93
Java 93
toolbar 93
search WSDL 496
select classifier 83
select generation target
preview 89, 410
self call message 12
self message 12
self return message 12
Self-Destroy
message (collaboration diagram) 216
message (sequence diagram) 239, 244
sender 236
sequence diagram 6
activation 236
actor 227
convert to collaboration diagram 226
create 225
merge models 225
message 236
object 231
object lifeline 231
OOM 224
UML 224
sequence number 213
create in collaboration diagram 219
decrease in collaboration diagram 222
increase in collaboration diagram 221
insert 221
instance link (collaboration diagram) 207
message (collaboration diagram) 213, 215, 546
message (sequence diagram) 238, 546
move 219
serialization class 479
Web service class 480
servlet 445
class 446
component 445
create 446
create from class diagram 446
create from selected class 446
generate 448
generate deployment descriptor 451

servlet (*continued*)
initialization 448
reverse engineering 452
synchronization 448
type 445
Web service 476
wizard 446
servlet class
servlet component 445
servlet component
servlet type 445
servlet type
genericservlet 446
httpservlet 446
user-defined 446
servlet Web service
generate 491
Setter
operation 111
severity
error 516
warning 516
share
object language 22
shortcut code uniqueness 526
show classes as data type 28
show direction
association (use case diagram) 53
show symbol
actor 48, 201, 229
signature
operation 542
simple data type 664, 678
SOAP 471
SOAP extension class 479
Web service operation 481
sorted association (class diagram) 141
source of association (use case diagram) 51
source of transition (activity diagram) 333
source of transition (statechart diagram) 279
specification
implementation operation 135
SQL queries 567, 572
SRU
generate 640
generate file 641
PowerBuilder 641
standard check
domain (OOM) 165
list of values 182
standard component
Web service 476
standard data type generation 664, 665, 680
standard Web service
generate 491
start (activity diagram) 303
check model 553
code uniqueness 554
composite activity 551
create 304
display preferences 305
missing transition 554
name uniqueness 554
properties 303
Rose import 585
tool 14
start (statechart diagram) 261
check model 553
code uniqueness 554
composite state 548
create 262
display preferences 263
missing transition 554
name uniqueness 554
properties 261
Rose import 583
tool 13
state 264
action 265
action order 548
change to atomic 264
change to composite 268
check model 547
class 510
classifier 264
code uniqueness 547
component 510
composite 264, 266
create 267
deferred events 265
display preferences 271
drag and drop in composite state 266
input transition 547
name uniqueness 547
properties 264
Rose import 583
tool 13
use case 510
state machine
UML 256

statechart diagram 6, 256
action 272
convert to composite state 259
create 259
default classifier 258
end 294
event 285
junction point 291
OOM 256
parameter 286
Rose import 582
start 261
state 264
sub-statechart diagram 257
transition 278
UML 256
stateless session bean
Web service 476
Web service generation 491
static
operation 120
static initializer 132
add 132
operation 132
stereotype
association (class diagram) 139
class 68
component 357
dependency (class diagram) 157
dependency (use case diagram) 60
EJB 423
generalization (class diagram) 152
generalization (component diagram) 370
implementation 152, 370
implicit 139
interface 98
node association 396
operation 121
organization unit (activity diagram) 323
package 32
realization 161
Rose import 575
XML-Schema 646
storedFunction 121
storedProcedure 121
sub-activity diagram 300
sub-class
generalization 151
sub-package
hierarchy 32
sub-statechart diagram 257
success
transition (activity diagram) 334
transition (statechart diagram) 280
super class 181
generalization 151
support delay
message (collaboration diagram) 217
message (sequence diagram) 30, 241
swimlane (activity diagram)
attach activity 327
change symbol format 330
copy and paste 329
create 323, 324
delete 330
detach activity 327
display horizontal 331
display vertical 331
move 329
organization unit 321, 328
resize 329
select symbol 328
UML 321
symbol format dialog box 330
synchronization
EJB 418
servlet 448
synchronization (activity diagram) 344
check model 555
code uniqueness 555
create 346
horizontal symbol 347
name uniqueness 555
properties 345
timeout 345
tool 14
transition 555
type 345
vertical symbol 347
synchronize
code editor 604
generated file 604
model 604
synonym
reverse engineer 590
syntax
variable 94

T

table
  generate 688
  prefix 688
target model 652
target namespace
  Web service component 475
  Web service component instance 483
templatecontent
  ASP.NET 464
  JSP 455
timeout
  synchronization (activity diagram) 345
  transition (activity diagram) 334
  transition (statechart diagram) 280
tModel 471
tool palette
  OOM 10
trace mode 620
transaction
  component 357
  EJB 412
transaction type 412
transition
  check model 555
  destination 555
  duplicated occurrence 556
  missing condition 556
  source 555
transition (activity diagram) 332
  alias 334
  condition 334, 339
  create 335
  destination 333
  display preferences 337
  exception 334
  flow type 333, 334
  incomplete decision 552
  incomplete synchronization 555
  missing on end 554
  missing on start 554
  properties 333
  Rose import 585
  source 333
  success 334
  timeout 334
  tool 14
  useless condition 556
transition (statechart diagram) 278
  alias 280
  condition 280
  create 282
  destination 279
  display preferences 284
  event arguments 281
  exception 280
  flow type 279, 280
  link to event 278
  merge model 278
  missing on end 554
  missing on start 554
  operation 281
  operation arguments 281
  properties 279
  Rose import 583, 584
  source 279
  success 280
  timeout 280
  tool 13
  trigger action 281
  trigger event 281
translate
  data type 663, 676
  PDM objects 675
trigger action
  transition (statechart diagram) 278, 281
trigger event 272
  action 272, 273, 274
  create 274
  event 285
  transition (statechart diagram) 278, 281
type
  business rule (OOM) 697
  component 358
  decision (activity diagram) 340
  synchronization (activity diagram) 345

U

UDDI 471
  extended attributes 475
UDDI operator URL 496
UDDI version 496
UML 3
  activity diagram 298
  class diagram 64
  collaboration diagram 196

UML (*continued*)
component diagram 354
object diagram 174
OMG xi, 3
sequence diagram 224
specification xi
state machine 256
statechart diagram 256
swimlane 321
terminology xi, 3
use case diagram 36
version xi
UML notation 306
undefined message (sequence diagram) 242
undefined recursive message (sequence diagram) 247
unordered association (class diagram) 141
unspecified trigger event
action of a state 548
unused event 550
update
CDM 672
OOM 659
PDM 688, 692
update class diagram of component 364
use case 38
action step 39
change to state 510
check model 527
class implementation 505
classifier 77
code uniqueness 527
copy and paste in statechart diagram 264, 510
create 41
display preferences 43
drag and drop in statechart diagram 264, 510
exception 39
extension point 39
implementation class 39, 506
interface 506
isolated 527
name uniqueness 527
post-condition 39
pre-condition 39
properties 38
related diagrams 41
Rose import 576
state 510
tool 10
use case diagram 4, 36
actor 44
association 50
create 37
dependency 59
generalization 54
OOM 36
UML 36
use case 38
use data type full name 29
use external WSDL
Web service component 474
Web service component instance 482
use fully qualified name 83

V

validate
OOM 516
value
attribute value 182
variable
format 94
justify 94
syntax 94
VB.NET
generate 620, 643
VB.NET generation option
generate code in .ASMX 486, 643
VB.NET generation task
compile source files 643
generate Web service proxy class 643
version
EJB 2.0 408
OOM 24
vertical
swimlane 331
visibility 528
association (class diagram) 140, 142
attribute (class diagram) 105, 107, 114
class 66, 69
generalization (class diagram) 152
generalization (component diagram) 370
generalization (use case diagram) 55
interface 97, 98
operation 120, 122, 138

W

warning
  OOM 516, 522
  severity 516
warning message
  display 520
Web service 470
  class diagram 480
  component 357
  component diagram 473
  component instance 482
  create 476
  create from class diagram 476
  create from selected class 476
  data type 478
  deploy in .NET 486
  deployment descriptor 485
  deployment diagram 482
  generate client side code 485
  generate EAR 485
  generate in .NET 486
  generate JAR 485
  generate WAR 485
  generation 490
  implementation 471
  implementation class 476
  namespace 476
  node 484
  OOM 470
  preview 475
  properties 473
  reverse engineering 493
  type 474, 476
  wizard 476
  XSD data type 478
Web service class 474
  preview 480
  properties 480
Web service interface 471
Web service method 120
Web service operation
  properties 481
Web service parameter
  properties 481
Web service proxy code
  .NET generation tasks 486
web.xml 485
WebLogic
  extended model definition 436
wizard
  create a component 365
  create a JSP 455
  create a servlet 446
  create an ASP.NET 464
  create an EJB 412
  create Web service 476
WSDL 471
  generation 620
  implementation 471
  import 493
  interface 471
  reverse engineering 493
  reverse options 494
WSDL comment
  Web service component 475
WSDL data type 478
  modify 479
  parameter (class diagram) 123
  Web method parameter 481
  Web service operation 481
WSDL editor
  user-defined button 475, 483
  Web service component 475
WSDL extended model definition
  .NET 21
  Java 21
WSDL URL
  import WSDL 493
  Web service component 474
  Web service component instance 482
WSDL2Local 478

X

XML
  generate 647
  objects 613
  reverse engineering 613, 615
XML generation 620
XML reverse engineering
  options 614
XML-Data
  generate 647
XML-Data generation option
  generate collections 647
XML-Data option
  generate 647

XML-DTD
generate 646
XML-Schema
generate 646
stereotype 646

Z

zip
reverse engineering Java 598
zoom
Rose import 575